UNIVERSIDAD AUTONOMA DE MADRID

ESCUELA POLITECNICA SUPERIOR

Grado en Ingeniería Informática

TRABAJO FIN DE GRADO

Diseño y análisis de videojuegos con orientación educativa

Sergio Martín García

Tutor: Sacha Gómez Moñivas

Mayo 2015

Resumen

Cada día, el crecimiento de dispositivos móviles es más importante y se refleja en los distintos lugares que necesitan de sus servicios para llevar una vida más cómoda. Es normal que todo el mundo se muestre interesado por la amplitud de posibilidades que ofrece un dispositivo móvil. Uno de los sectores que se están beneficiando de estas mejoras es el sector educativo. Gracias a las nuevas tecnologías, los estudiantes tienen más facilidad de acceso autónomo. Sin embargo, no todas las materias están utilizando estos nuevos medios disponibles para el aprendizaje de los estudiantes.

El presente proyecto viene a aportar nuevas formas de aprendizaje en la educación a través de videojuegos en el ámbito histórico. Con este proyecto se pretende enseñar a los estudiantes algunas nociones básicas acerca de las épocas históricas, los materiales utilizados en las mismas y poder realizar construcciones con bloques creados por el estudiante. También la aplicación permitirá al profesor seguir el progreso del estudiante.

Este proyecto pretende ser la idea inicial de orientación educativa para las materias históricas. Gracias a los juegos de este tipo, los estudiantes aprenden de forma más dinámica, divertida y menos monótona.

El siguiente documento presenta toda la información necesaria para comprender el trabajo realizado en el proyecto. Se comenzará con una breve descripción del juego desarrollado, con los objetivos y retos propuestos, y se analizarán cada una de las fases por las que ha pasado la aplicación, desde el estudio inicial hasta las pruebas realizadas.

Por último, puede encontrarse un manual de uso para el estudiante y otro manual de uso para el profesor.

Palabras clave

Android, Aplicación, Videojuego, Aprendizaje, Época, Bloque, Textura, Construcciones

Abstract

Each day, the growth of these devices is more important and this is reflected in the different places that need their services to achieve a more comfortable life.

It is normal that everyone shows interest in the wide range of possibilities offered by a mobile device. One of the sectors that is benefiting from these improvements is the educational sector. Thanks to new technologies, students have easier autonomous access. However, not all subjects are using these new means available for student learning.

This project is to provide new ways of learning in education through games in the historic area. This project aims to teach students some basic notions about historical periods, the materials used at the time and be able to constructions with blocks created by students. The application also allows the teacher to follow the student's progress.

This project aims to be the initial idea of an educational guidance for history subject. Thanks to these games, students will learn in a more dynamic, entertaining and less monotonous way.

The following document presents all the information necessary to understand the work done in the project. It will begin with a brief description of the game developed, taking into account the proposed objectives and challenges, and an analysis of each of the developing phases that the application has experienced, from the initial study to the tests carried out.

Finally, you may find a user manual for the student and one manual for the teacher.

Keywords

Android, Application, Videogame, Learning, Age, Block, Texture, Building

Glosario

Actividad: Cada una de las pantallas de la aplicación móvil.

Android: Sistema operativo basado en Linux para dispositivos móviles.

Api (*Application Programming Interface*): Conjunto de subrutinas, funciones y procedimientos que ofrece cierta biblioteca para ser utilizado por otro software como una capa de abstracción.

App: Abreviatura de aplicación.

Bitmap: Formato de imagen de mapa de bits.

Cliente: Aplicación informática que consume un servicio remoto de otro ordenador conocido como servidor.

HTTP (*Hypertext Transfer Protocol*): Protocolo usado en cada transacción de la web. Sigue el esquema petición-respuesta entre un cliente y un servidor. No guarda ninguna información sobre conexiones anteriores.

Kernel: Software que constituye una parte fundamental de un sistema operativo.

Layout: Tipo de panel utilizado en Android.

Script: Archivo de ordenes simple almacenados en un archivo de texto plano.

Servidor: Aplicación en ejecución capaz de atender las peticiones de un cliente y devolverle una respuesta en concordancia.

Smartphone: Anglicismo. Teléfono móvil construido sobre una plataforma informática móvil que realiza actividades asemejándose a una computadora.

Agradecimientos

Después de haber terminado mi Trabajo de Fin Grado, quiero dar las gracias a todos los que lo han hecho posible:

A Sacha, gracias por haberme permitido realizar este proyecto y por su ayuda recibida en tantas tutorías.

A Pablo, gracias por haberme ayudado y guiado tanto de manera desinteresada.

Y sobre todo, gracias a mis amigos y mi familia por su apoyo en los buenos y malos momentos.

ÍNDICE DE CONTENIDOS

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

1. INTRODUCCIÓN

A medida que han ido transcurriendo los años, las nuevas tecnologías han ido obteniendo una gran importancia en todos los ámbitos de la vida, desde el ámbito profesional hasta el personal. Una prueba de ello son los llamados *smartphones*. Gracias a este invento, hoy en día todo el mundo tiene una minicomputadora en la palma de su mano, que ofrece infinidad de ventajas para conseguir facilitarnos la vida.

Día a día se desarrollan distintas aplicaciones para *smartphones* que van desde poder acceder a la cuenta de tu banco y ver tus movimientos hasta conseguir comunicarte con varias personas a través de mensajería instantánea en tiempo real.

Entre las ventajas del *smartphone* se puede hablar de la importancia que ha tenido este invento en el sector educativo, ya que cada vez está más presente para poder ayudar a los estudiantes a alcanzar sus objetivos.

Precisamente esto último, es lo que ha impulsado a la realización de este proyecto en donde poder facilitar a los estudiantes, en el campo de la arquitectura y la historia, su proceso de aprendizaje.

1.1 Motivación

En décadas anteriores todo se realizaba con herramientas tan simples como son el lápiz y el papel, destacando la máquina de escribir como uno de los inventos más sobresalientes de esa etapa. Con el paso del tiempo, empezaron a emerger numerosas ideas que cambiarían por completo la forma de vivir de aquel entonces. Gracias a este cambio de visión apareció el primer ordenador personal “Olivetti Programma 101” en 1965 y el primer teléfono móvil “Ericsson NMT450” en 1980.

En 1994, se juntó la inteligencia básica de un ordenador y las llamadas inalámbricas de un teléfono móvil y gracias a ello nació el Ericsson R380, el primer teléfono inteligente (“Smartphone”) [1].

En la actualidad, los smartphone han avanzado de forma notoria, aunque tengan el mismo espíritu de sus comienzos. Debido a esta evolución, las personas podemos realizar múltiples funciones que facilitan la vida cotidiana de las personas (organizar nuestra agenda, estar informado en todo momento de lo que está pasando en el mundo,..).

La motivación principal de este proyecto es conseguir, gracias a una aplicación creada para un “smartphone”, que sirva como objeto de enseñanza para alumnos de un colegio, instituto o universidad. De esta manera, se conseguirán mejores resultados por parte de los alumnos si aprenden de una forma más dinámica, divertida y menos monótona.

1.2 Objetivos del Proyecto

El objetivo de este proyecto es realizar una aplicación *smartphone* con orientación educativa para la rama histórica. Esta aplicación tendrá los siguientes contenidos:

- El objetivo principal es conseguir crear distintas construcciones con bloques creados por el usuario y hechos de determinados materiales.

- Estará compuesta por distintas épocas históricas que tendrán los distintos materiales utilizados en esas épocas.

- Estará pensada para que cualquier profesor pueda añadir otras épocas con materiales, sin que tengan nociones básicas de informática.

- El usuario tendrá distintas herramientas para poder crear los distintos bloques de los que se compondrán las construcciones.

Otro de los principales objetivos que se quiere conseguir es que se trate de una aplicación con una interfaz fácil y sencilla, donde el usuario sepa utilizarla sin ningún problema.

Si a estos objetivos le añadimos la dificultad de crear un proyecto así desde cero, donde tú decides tus objetivos, tus estilos y de que se compondrá, produce un desafío muy interesante donde saber aplicar gran parte de lo aprendido en la universidad.

1.3 Fases del proyecto

En este punto se definirán los pasos por los que tiene que pasar el proyecto para poder llegar a la aplicación final que podrá utilizar el estudiante, que son las fases tradicionalmente utilizadas en un proyecto software:

- **Estudio**: Investigación de las tecnologías que se utilizarán así como una comparación de las otras aplicaciones que tienen estos objetivos comunes.
- **Análisis**: En esta fase se definirán los requisitos principales que indicarán como tiene que funcionar la aplicación. Estos requisitos marcarán los objetivos necesarios para que el juego tenga toda su funcionalidad.
- **Diseño**: En este punto se diseñará la arquitectura de la que se compondrá la aplicación. Esto implica el diseño de la base de datos, clases que se utilizarán y el diseño de la navegación a través de la interfaz. También se crearán los diagramas necesarios para indicar estos puntos a través de esquemas.
- **Implementación**: Una vez que se han definido los requisitos y el diseño que se va a seguir, se pasa a la implementación del código que cumplirá los objetivos requeridos para que todo funcione correctamente.

- **Pruebas**: Se realizarán todo tipo de pruebas para comprobar que la implementación ha sido correcta y que no habrá ningún caso donde el usuario, que utilice la aplicación, no vaya a obtener un resultado incorrecto a lo esperado.
- **Documentación**: Una vez que todo funciona correctamente, se deberá explicar de forma clara y concisa a través de documentos que expliquen todo lo que implica al proyecto. Este punto es indispensable para después poder realizar un mantenimiento adecuado.
- **Mantenimiento**: Aunque no lo parezca, este es el punto que supone casi un 80% del ciclo de la vida del software [2]. Esta fase puede que no se realice de forma tan exhaustiva como en otros proyectos, dado que no se sabe de la continuidad del mismo en un futuro próximo.

Todas estas fases están presentes en el ciclo de vida utilizado en este proyecto. El ciclo de vida utilizado es del tipo cascada. Este modelo se caracteriza en que para poder acceder a la siguiente fase se debe de haber completado la fase anterior, así garantizas un orden concreto que seguir. Se ha elegido este modelo ya que a medida que se va desarrollando el software, se puede observar el progreso del mismo en distintos puntos del ciclo de vida. Sin embargo, se ha tenido que acceder a fases terminadas anteriormente para realizar algunos cambios dado que al comienzo del proyecto los requisitos no estaban completos al desconocer lo que abarcaría la aplicación.

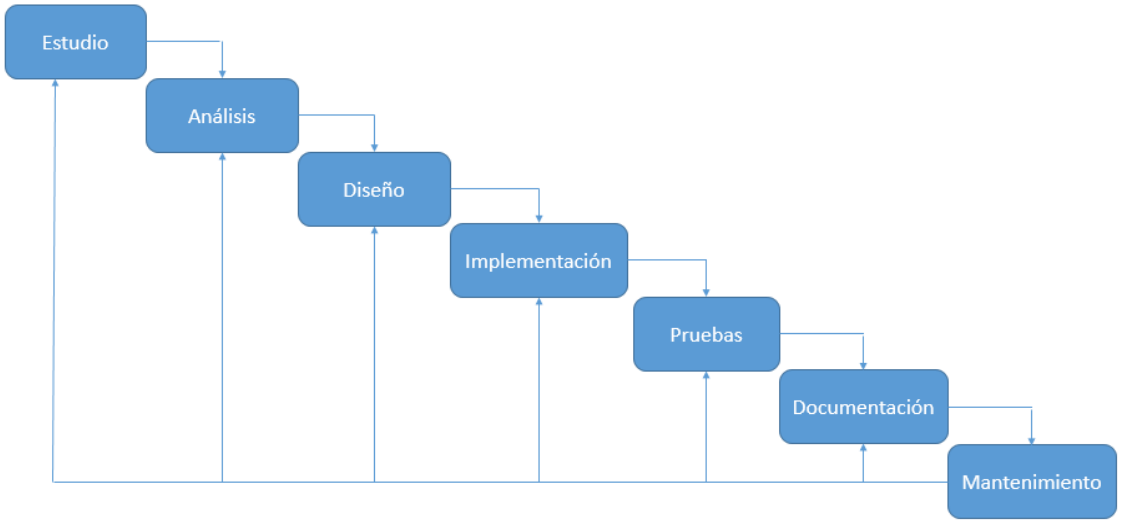

Ilustración 1. Ciclo de vida cascada

1.4 Estructura del documento

Este documento recoge todas las fases mencionadas anteriormente. Para cada una de ellas se ha reservado un capítulo donde se desarrollará de forma más amplia y se comentarán todos los puntos clave que la componen. También contiene un manual de usuario, con el que el cliente podrá utilizar la aplicación sin ningún problema, y un manual de usuario para profesores, que podrán utilizar para añadir sus texturas y épocas.

En *2.Estado del arte* se expondrán los conceptos e intereses básicos de Android, el trabajo de investigación previo al comienzo del proyecto y las herramientas utilizadas para poder crear la aplicación.

Después en *3.Análisis* se analizarán todos los requisitos funcionales y no funcionales que componen al proyecto.

A continuación, en *4.Diseño* se documentarán todas las decisiones de diseño tomadas. Se comentará la arquitectura utilizada, el diseño de la interfaz, el diseño del servidor y el diseño de la estructura de datos.

En *5.Implementación* se desarrollará todo lo relacionado con la implementación de la aplicación, tanto por la parte del cliente como la parte del servidor. Se pasarán a analizar los módulos utilizados y los procedimientos utilizados.

El capítulo *6.Pruebas* mostrará todas las pruebas realizadas en la aplicación para evitar que el usuario obtenga errores no deseados.

2. ESTADO DEL ARTE

En este capítulo se ha llevado a cabo un breve estudio acerca de Android donde se detallan sus características y su estructura. También se han comentado aquellos aspectos referentes a la investigación previa al proyecto para poder elegir aquellas plataformas que sacarían más rendimiento a la aplicación.

Por último se relatan aquellas herramientas utilizadas para poder desarrollar todo el proyecto.

2.1 Android

Android es un sistema operativo basado en el *kernel* de Linux y que ha sido diseñado principalmente para uso de *tablets* y *smartphones*, aunque en la actualidad se está utilizando también para relojes inteligentes, televisiones y automóviles.

Inicialmente fue desarrollado por Android Inc., empresa que Google respaldó económicamente y que en 2005 compró. El primer móvil con el sistema operativo Android fue el "HTC Dream" y se vendió en octubre de 2008. Los dispositivos de Android venden más que las ventas combinadas de Windows Phone y IOS como podemos observar en la *Ilustración 2*.

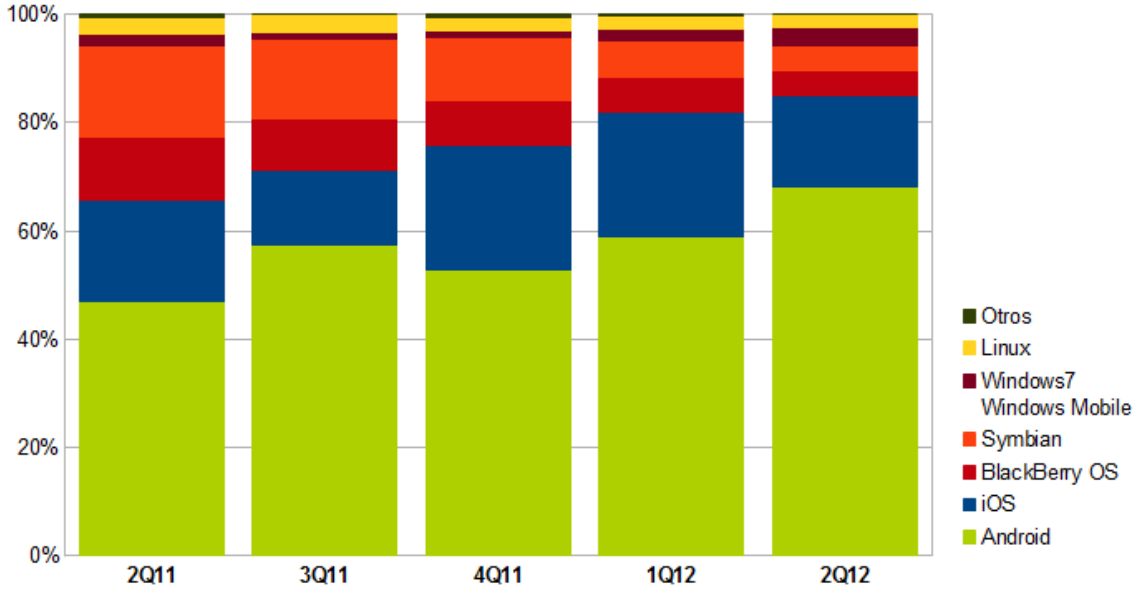

Ilustración 2. Ventas de sistemas operativos en 2011-2012

El SDK (*Software Development Kit*) de Android proporciona todas las herramientas necesarias para poder desarrollar aplicaciones utilizando Java. Al tratarse de un software libre, los usuarios han ido aplicando sus mejoras al sistema [3].

2.1.1 Estructura de Android

Como se puede observar en la *Ilustración 3* la arquitectura de Android es la siguiente:

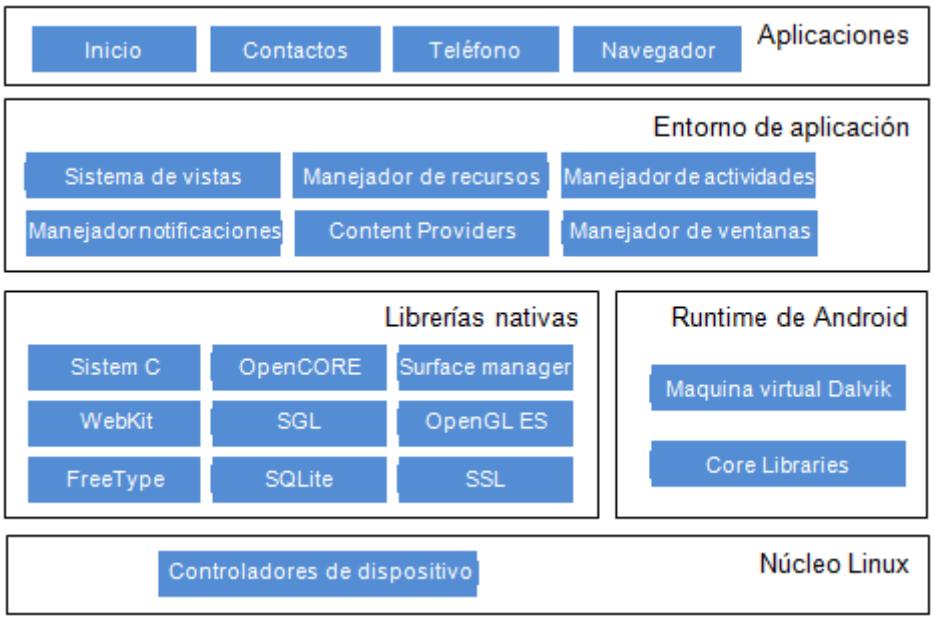

Ilustración 3. Arquitectura de Android

El **núcleo Linux** está basado en el *kernel* del sistema operativo Linux versión 2.6. Esta capa proporciona manejo de memoria, servicios de seguridad, multiproceso y soporte para controladores de dispositivo.

El **Runtime** se compone de librerías base que proporcionan la mayor parte de las funciones disponibles de las bibliotecas base del lenguaje Java. Cada aplicación Android corre su propio proceso, con su propia instancia de la máquina virtual *Dalvik*.

En el mismo nivel, podemos encontrar las **librerías nativas**, escritas en C/C++ y compiladas con el código nativo del procesador. Estas características se exponen a los desarrolladores a través del marco de trabajo de aplicaciones de Android.

El **marco de trabajo de aplicaciones** ofrece una plataforma de desarrollo libre de aplicaciones. De esta manera, los desarrolladores pueden utilizar los mismos APIs del *framework* usados por las aplicaciones base. Como se puede observar, hay controladores de notificaciones, recursos, ventanas y actividades.

Por último está la **capa de aplicaciones** formada por el conjunto de aplicaciones que tenga el dispositivo. Esta capa tiene una serie de aplicaciones base como son un cliente de correo electrónico, calendario, mapas, navegador, contactos y otros.

2.1.2 Versiones de Android

Curiosamente, las versiones de Android reciben del inglés el nombre de diferentes postres. Además estas versiones están ordenadas por orden alfabético. Estos detalles se pueden apreciar en la *Tabla 1* [3].

Letra	Nombre	Traducción	Versión
A	Apple Pie	Tarta de manzana	1.0
B	Banana Bread	Pan de plátano	1.1
C	Cupcake	Cupcake	1.5
D	Donut	Dónut	1.6
E	Éclair	Pepito	2.0/2.1
F	Froyo	Yogur Helado	2.2
G	Gingerbread	Pan de jengibre	2.3
H	HoneyComb	Panal de miel	3.0/3.1/3.2
I	Icre Cream Sandwich	Sandwich de helado	4.0
J	Jelly Bean	Gragea	4.1/4.2/4.3
K	Kit Kat	Kit Kat	4.4
L	Lollipop	Piruleta	5.0/5.1/5.1.1
M	Muffin (Rumor)	Muffin	6.0/6.1/6.1.1

Tabla 1. Nombre de las versiones de Android

No todo el mundo tiene actualmente la última versión lanzada por Android en su dispositivo móvil. En la *Tabla 2* se pueden observar el porcentaje de terminales que usan las distintas versiones [3].

Versión	Nombre	Nivel API	Cuota
5.1	Lollipop	22	0,7%
5.0	Lollipop	21	9,0%
4.4	Kit Kat	19	39.8%
4.3	Jelly Bean	18	5,5%
4.2	Jelly Bean	17	18,1%
4.1	Jelly Bean	16	15,6%
4.0	Ice Cream Sandwich	14	5,3%
2.3	Gingerbread	10	5,7%
2.2	Froyo	8	0,3%

Tabla 2. Cuota de las versiones

La aplicación a desarrollar en este proyecto está enfocada para dispositivos de Android 4.0 o superior. En caso de no tener una versión adecuada, puede que la aplicación no tenga toda la funcionalidad desarrollada o incluso que no pueda ni ser ejecutada en el dispositivo. Se ha utilizado un “Asus Memo Pad 10”, con Android 4.2.2, como dispositivo de pruebas de la aplicación.

2.2 Trabajo previo de investigación

Se ha demostrado que los videojuegos con orientación educativa motivan más a los estudiantes que los métodos tradicionales [4], produciendo así mejores resultados. Esta fue la razón por la que se ha realizado este proyecto.

En un primer momento se dudó sobre en qué plataforma debía ser desarrollado el proyecto, como web, como videojuego para ordenadores o como aplicación para dispositivos móviles.

La elección elegida fue realizar una aplicación móvil, ya que es un mercado que está en auge continuamente y ofrece posibilidades de interacción que no permite un ordenador.

En el mercado de los sistemas operativos móviles hay dos pioneros que destacan por encima de los demás: Android e IOS. Como se pudo observar en la *Ilustración 2*, Android supera con creces a los demás sistemas operativos activos. Esto ha provocado a su vez que se desarrollen más aplicaciones para Android que para las otras opciones como se puede observar en la *Ilustración 4*.

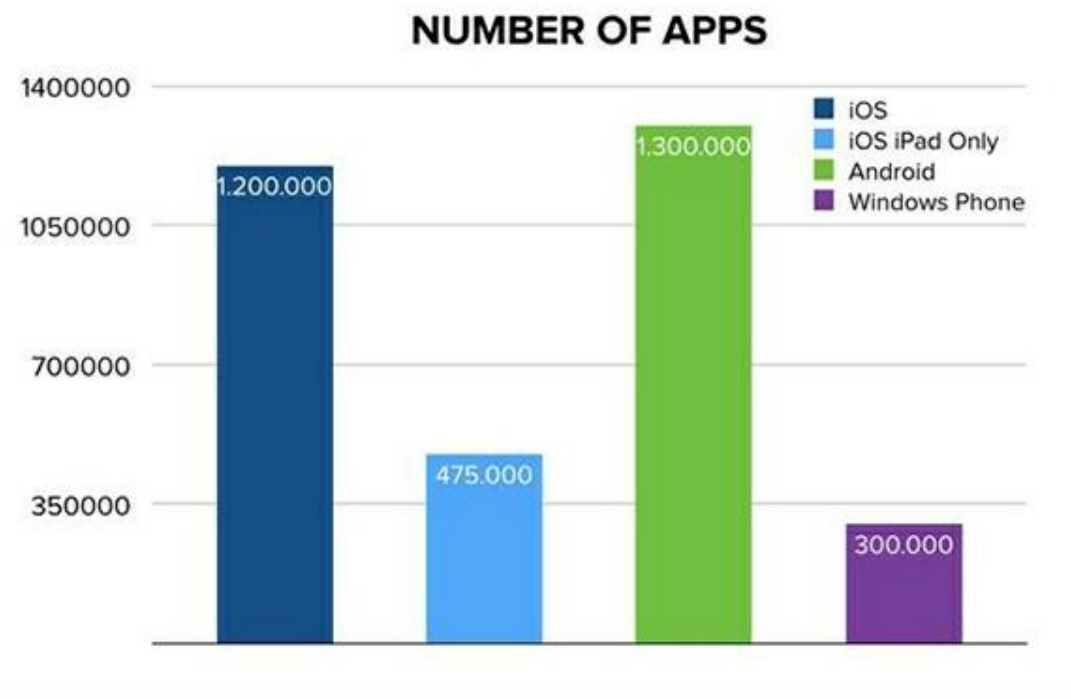

Ilustración 4. Número de aplicaciones disponibles en Agosto 2014

Dentro del ámbito de la educación, hay múltiples materias que la engloban. Sin embargo, en el caso de las aplicaciones móviles, la materia que está más presente es la lengua, con juegos como *Apalabrados* o *Ruzzle.* También existen juegos que engloban a varias temáticas como el arte, la literatura o la geografía entre otros (*Triviados* y *Preguntados*).

Sin embargo, no se observa un gran número de aplicaciones dedicadas exclusivamente a la educación histórica. Esta temática tiene un amplio abanico de recursos y posibilidades. Por estas razones, la historia ha sido la temática elegida para la aplicación.

Ya se ha trabajado anteriormente con esta temática en distintos juegos de ordenador como la saga "*Imperivm*", donde recreabas las grandes batallas que hubo en la historia. También está la saga "*Age of Empires*", donde el jugador se encargaba de crear su propia civilización desde las primeras aldeas hasta los fuertes castillos gracias a los distintos materiales que podían recolectar.

Ilustración 5. Imperivm y Age of Empires

2.4 Herramientas y tecnologías utilizadas

En este punto se describirán las herramientas utilizadas durante el desarrollo del proyecto software:

- **Eclipse SDK**: Programa informático que se compone de varias herramientas de programación de código para desarrollar las aplicaciones software. Se ha utilizado el SDK de Android para poder programar la aplicación móvil.
- **PGAdmin**: Herramienta de código abierto para la administración de bases de datos PostgreSQL que contiene herramientas de consulta SQL y una interfaz que ofrece todos los datos necesarios de una base de datos. Se ha utilizado para crear, modificar y visualizar los datos que contiene el servidor.
- **Filezilla**: Cliente FTP multiplataforma de código abierto y software libre. Gracias a este programa, se ha podido interactuar con el servidor para poder añadir y obtener las imágenes que están presentes en el mismo.
- **Microsoft Office**: Se trata de una Suite Ofimática con distintas aplicaciones de escritorio.
- **Dropbox**: Servicio de alojamiento de archivos multiplataforma en la nube. Muy útil para poder acceder a tus archivos desde distintos ordenadores.
- **Paint.NET**: Editor gráfico más potente y complejo que Paint pero al mismo tiempo fácil de usar. Este editor se ha utilizado para algunos estilos aplicados en la aplicación, por ejemplo la serigrafía del título de la aplicación.

3. ANÁLISIS

En este apartado se va a presentar la fase de análisis de un proyecto, cuyo principal objetivo es la definición de los requisitos funcionales y no funcionales del proyecto. Este punto es de mucha importancia, ya que el contenido de la aplicación y el funcionamiento del mismo dependerán de los requisitos elegidos.

Los requisitos son necesidades documentadas del comportamiento esperado por un producto. Se dan dos tipos de requisitos: **Funcionales** y **No Funcionales**.

Un requisito funcional define una función del sistema de software o de sus componentes. Mientras que un requisito no funcional completa a los requisitos funcionales y se enfocan en el diseño o la implementación [5].

3.1 Requisitos funcionales

Se dividirán en subcategorías, para tener los requisitos de manera organizada, y se dará una explicación acerca de la labor de cada uno de los requisitos mostrados.

3.1.1 Gestión de épocas

- **RF01 Obtener épocas**
  Se conectará a la base de datos del servidor donde realizará una consulta de todas las épocas disponibles para poder mostrarlas por pantalla. Estas épocas se irán cargando en segundo plano para que el usuario pueda elegir una época que ya está cargada, sin tener que esperar a que se carguen las demás.

- **RF02 Mostrar épocas**
  Una vez que se han obtenido las épocas, se mostrarán en la pantalla en forma de imágenes que describan a la misma. Estas imágenes se obtendrán también de la consulta realizada para obtener las épocas disponibles. Como se comentó en RF01, las imágenes se irán cargando en segundo plano.

- **RF03 Mostrar descripción de la época**
  Si el usuario selecciona una imagen de una época, el sistema devolverá un diálogo con una descripción simple de la misma. Del mismo modo que RF01 y RF02, esta información se obtendrá de la consulta utilizada en los anteriores requisitos.

- **RF04 Elección época**
  El diálogo que se mostrará con la descripción de la época contiene también un botón que te ofrecerá la posibilidad de elegir esa época. Según la época que se seleccione, los materiales para los bloques irán cambiando.

3.1.2 Gestión de bloques

- **RF05 Carga texturas de la época**
  Se conectará a la base de datos del servidor y se realizará una consulta, con la época elegida en el menú selector de épocas, que devolverá todos los materiales existentes en la base de datos para la época elegida. Estos materiales se cargarán en segundo plano y se mostrarán en pantalla en forma de imágenes.

- **RF06 Mostrar paneles en la pantalla**
  El menú de edición de bloques estará dividido en distintos paneles que tendrán distintas funcionalidades. Se dividirá la pantalla en los siguientes paneles:
  - Panel con las texturas de la época
  - Panel del modo edición con tres sub-paneles:
    - Panel donde se encuentre el bloque que se creará
    - Panel donde se encuentren los botones para realizar cambios en el bloque
    - Panel donde se encuentre los datos referentes a su tamaño y tipo de material

  Por último, tendrá un panel final donde se encontrarán todos los bloques creados.

- **RF07 Cambiar tamaño del bloque**
  El usuario podrá cambiar el tamaño del bloque con unos editables de la altura y anchura del bloque (en pixeles). Estos editables mostrarán al principio el tamaño en pixeles del bloque por defecto. El usuario podrá cambiar la altura y la anchura con esos editables, introduciendo otros pixeles. Finalmente, se pulsará un botón para realizar los cambios.

- **RF08 Rotar el bloque**
  El usuario dispondrá de un botón para rotar el bloque que se está creando, incluso con los cambios que ya tenga realizados en el mismo.

- **RF09 Deshacer cambios en el bloque**
  Se podrán deshacer todos los cambios realizados en el bloque, para volver al bloque por defecto y comenzar de nuevo.

- **RF10 Cortar manualmente el bloque**
  El usuario tendrá la posibilidad de cortar el bloque manualmente. Para poder realizar esta acción, el usuario deberá trazar una línea recta vertical en el bloque por defecto con su dedo. Esta línea deberá ser trazada de abajo a arriba. Si se ha realizado correctamente, el sistema detectará el corte y se quedará con la parte más grande de los dos trozos restantes.

- **RF11 Cambiar textura del bloque**
  El usuario podrá cambiar el cualquier momento la textura del bloque que está editando. Para ello deberá seleccionar una de las imágenes de las texturas y automáticamente cambiará la imagen del bloque por defecto.

- **RF12 Crear el bloque**
  Una vez que se hayan realizado todos los cambios pertinentes (o ningún cambio, si así lo desea el usuario), se usará un botón para crear el bloque editado. Dicho bloque aparecerá en el panel de bloques creados.

- **RF13 Mostrar todos los bloques creados**
  Habrá un panel con todos los bloques creados previamente por el usuario. Estos bloques aparecerán con una imagen de cómo ha quedado el bloque creado. Como el usuario puede haber creado múltiples bloques, se utilizará un sistema de botones que permitan ver los bloques por grupos. Es decir, se mostrará un número determinado de bloques en el panel y estos cambiarán con botones como "*Anterior*" o "*Siguiente*" para poder ver los restantes.

- **RF14 Reiniciar bloques**
  Si el usuario no está satisfecho con los bloques creados, podrá volver a comenzar de nuevo con un botón.

- **RF15 Comenzar partida**
  Cuando el usuario lo considere oportuno, podrá pulsar un botón que le permita acceder a la pantalla de juego con los bloques creados. Si el usuario no ha creado ningún bloque y pulsa dicho botón, el sistema le ofrecerá la posibilidad de crear automáticamente unos bloques por defecto.

3.1.3 Gestión de partida

- **RF16 Mostrar todos los bloques creados en el modo edición**
  El usuario podrá ver todos los bloques creados anteriormente en un panel superior. Si existen múltiples bloques, se creará un *scroll* horizontal que permita ver todos los bloques creados. Los bloques estarán acompañados de un nombre que contenga información del bloque (Ej. bloqAdobe1, se trata de un bloque de Adobe y fue el primer bloque creado).

- **RF17 Seleccionar bloque y dejarlo caer**
  El usuario tendrá que seleccionar un bloque creado para poder utilizarlo en su construcción. Para realizar esta acción, el usuario seleccionará el bloque que desee y, a continuación, pulsará en la posición de la pantalla donde desee que caiga. El sistema automáticamente prohibirá volver a seleccionar ese bloque, dado que ya ha salido utilizado.

- **RF18 Reiniciar Partida**
  El usuario podrá reiniciar la partida en cualquier momento a través de un botón.

- **RF19 Finalizar Partida**
  El usuario podrá finalizar la partida en cualquier momento aunque no haya colocado todos los bloques editados.

3.1.4 Simulación de física

- **RF20 Caída de bloques con simulación gravitatoria**
  Una vez que se ha elegido donde caerá el bloque en la pantalla, el bloque descenderá verticalmente simulando una caída con gravedad hasta que otro bloque o el final de la pantalla se lo impida.

- **RF21 Apilamiento de bloques en distintas situaciones**
  Mientras que el bloque está cayendo, pueden darse diversas situaciones en las que el bloque deberá parar de caer:

  - El bloque caerá sobre otro bloque y se quedará encima.
  - El bloque caerá sobre un bloque pero su punto de equilibrio puede provocar que realice una rotación contra ese bloque. Puede que mientras que rota, el bloque colisione con otro bloque.
  - El bloque cae entre dos bloques con un hueco suficiente como para que no caiga entre los dos y se sostenga.
  - El bloque llegará al final de la pantalla y parará.

- **RF22 Rotación de bloques**
  Puede que el bloque se encuentre con otro bloque pero caiga en una posición donde el punto de equilibro del bloque que estaba cayendo no sea correcto y el bloque tenga que caer lateralmente. En este momento, el bloque realizará una rotación de 90º con el bloque encontrado (o rotación de -90º si es hacia la izquierda).
  Puede que cuando el bloque rote, en su lateral haya más bloques quietos y tenga que reaccionar ante una situación de las mostradas en RF21.

- **RF23 Parar el juego si un bloque está mal colocado**
  Un bloque cae contra otro bloque y el punto de equilibrio del bloque que caía no es correcto, con lo que provoca una rotación. Sin embargo, mientras está rotando choca con otro bloque que estaba al lado del bloque con el que se encontró. Esto provoca que el bloque que descendía se quede inclinado entre los dos bloques (*Ilustración 29*). Esta situación no permite volver a poner bloques en la construcción. La solución que se ofrece es reiniciar la partida.

3.1.5 Gestión de la aplicación

- **RF24 Preferencias**
  Se podrá acceder a las preferencias a través de "*Opciones*" desde las pantallas de Selección de Época y en el Modo Edición. Todos los cambios realizados en las preferencias se mantendrán aunque se cierre la aplicación. Solo se borrarán si se desinstala la aplicación o si se borran los datos desde los ajustes.

- **RF25 Realizar captura de pantalla de la construcción**
  Cuando el usuario finalice una partida, el sistema se encargará de realizar una captura de pantalla de la construcción final que realizo el usuario. Esta captura le servirá al profesor para poder ver todos los progresos realizados por el estudiante.

- **RF26 Enviar correo**
  Una vez que el usuario pulse el botón encargado de finalizar la partida, el sistema le mostrará al usuario un correo relleno con información, creada por la aplicación, donde estará puesto el correo del profesor (que se indicará a través de una opción de las preferencias), el nombre del estudiante (otra opción que tendrá las preferencias), la captura de pantalla de la construcción y un mensaje que informa de todos estos datos. El usuario únicamente tendrá que pulsar el botón "*Enviar*" de su sistema de correo para enviar el mensaje a su profesor.

3.2 Requisitos no funcionales

- **RNF01 Usabilidad**
  El sistema mostrará una interfaz simple y atractiva que no provoque confusión al usuario. No es necesario que el usuario tenga conocimientos informáticos para poder utilizarlo, bastará con que este habituado al uso de aplicaciones simples en dispositivos móviles. Se ofrecerá una interfaz intuitiva para que en todo momento el usuario este informado de los movimientos que realiza.

- **RNF02 Mantenibilidad**
  El código está estructurado de manera que en el caso de que haya modificaciones o errores en un futuro, se solucionen fácilmente y realizando los cambios mínimos.
  El código será limpio, estructurado y argumentado con comentarios para saber en todo momento como funciona cada parte. Se han utilizado en todo momento nombres claros, que describan para que sirve cada elemento, y los recursos de Android (estilos, constantes, cadenas globales) para poder realizar cambios más fácilmente.

- **RNF03 Fiabilidad**
  Se controlará en todo momento posibles fallos que puedan provocar el cierre inesperado de la aplicación. De esta manera si hay, por ejemplo, un fallo de conexión al servidor no provocará un cierre de la aplicación

- **RNF04 Rendimiento**
  La aplicación irá siempre lo más rápido posible. Para facilitar esta labor, muchas tareas se han realizado en segundo plano para poder continuar mientras se cargan algunos datos. Estas tareas en segundo plano son peticiones al servidor pero al tratarse de un servidor externo, no se puede garantizar que vaya a gran velocidad siempre, dado que este servidor está siendo utilizado para otras labores externas a este proyecto. Sin embargo, todo lo externo al servidor, como cálculos matemáticos o comparación en bucles, se puede garantizar que se ha optimizado lo máximo posible para ofrecer la mejor experiencia al usuario.

- **RNF05 Documentación**
  Este mismo documento contiene un anexo con un manual de usuario para saber cómo utilizar la aplicación. También hay un anexo con un manual para el profesorado, por si quiere introducir sus propias épocas con sus propias texturas. Tal y como se mencionó en RNF02, la codificación de la aplicación esta comentada de forma ordenada y clara para poder saber en todo momento lo que significa cada método o variable.

4. DISEÑO

En este capítulo se va a presentar la fase de diseño de un proyecto donde se decide cual será la arquitectura del sistema y el diseño utilizado. Un diseño apropiado es una parte muy importante, ya que según el diseño elegido será más fácil organizar todo el proyecto y las modificaciones no resultarán complicadas.

En este punto se explicará la arquitectura elegida para la aplicación y diseño utilizado para la interfaz, la base de datos y el servidor.

4.1 Arquitectura de la aplicación

Se ha elegido un modelo cliente-servidor para la arquitectura del proyecto. Este modelo se basa en la idea de que un cliente envía peticiones a un servidor y este mismo le da respuesta.

Como se puede ver en la *Ilustración 6*, nuestro cliente es el dispositivo móvil, donde estará alojado la aplicación. Este cliente envía peticiones HTTP al servidor, el servidor procesa la petición y junto con la base de datos, envía la respuesta al cliente.

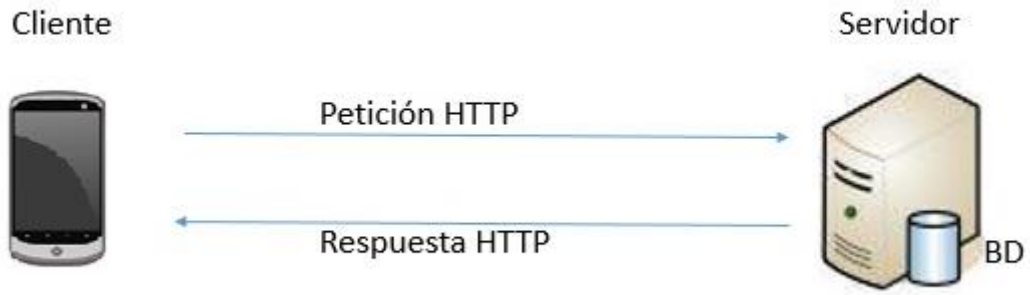

Ilustración 6. Modelo cliente-servidor

Era necesario tener un servidor con base de datos externo al dispositivo móvil, ya que en este sistema se almacenarán las imágenes y los datos correspondientes a las épocas y texturas utilizadas en la aplicación. Sin embargo, se ha utilizado también una base de datos local en el dispositivo móvil, que se explicará más adelante.

La principal ventaja de utilizar este modelo es que se pueden incrementar los recursos fácilmente, por lo que un profesor podría añadir otras épocas, imágenes de texturas u otra información a través de una página web básica, a la que pudiese acceder, y guardar automáticamente los datos en el servidor. De esta manera, cuando un usuario cargase su aplicación en el dispositivo móvil, automáticamente tendría todas las novedades actualizadas. De todos modos, no es la única ventaja que tiene este modelo ya que puede ser escalable y favorece la encapsulación.

La principal desventaja que puede tener este modelo es que puede tener problemas de congestión de tráfico cuando una gran cantidad de clientes envía peticiones simultáneas. Pero como este proyecto no está pensado para ser utilizado por una gran cantidad de usuarios al mismo tiempo, no es un problema grave. Si fuese necesario que lo utilizasen varios usuarios al mismo tiempo, se podría incorporar otro servidor para poder soportar más peticiones.

4.2 Diseño de la interfaz

Toda aplicación móvil tiene su parte de interfaz, con la que se accede a toda la funcionalidad codificada internamente.

Las aplicaciones en Android funcionan principalmente por actividades, donde cada actividad simula a una pantalla de la aplicación que puede ver el usuario. Estas actividades están formadas por la parte lógica y la parte gráfica. La parte gráfica es un XML que tiene todos los elementos que se ven en la pantalla, declarados con etiquetas parecidas al HTML. Es decir, el diseño de una página XML es parecido al diseño de una página web HTML.

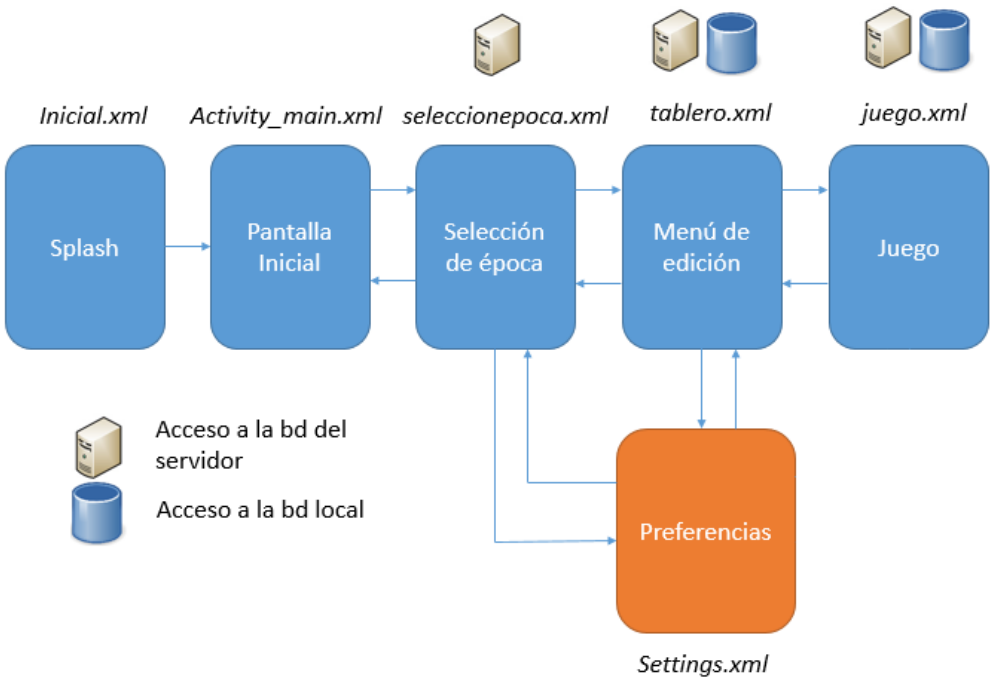

Ilustración 7. Mapa de navegación

En la *Ilustración 7* se puede observar el mapa de navegación de la aplicación. Los recuadros azules son cada una de las pantallas o actividades que tiene el juego. Encima de los recuadros, se encuentra la nomenclatura utilizada para cada una de esas pantallas.

El recuadro naranja simboliza a las preferencias utilizadas en la aplicación. Aunque también es una actividad, se ha querido dar otro color para poderla diferenciar del resto de pantallas. Las flechas dibujadas indican el recorrido que siguen las pantallas. De esta manera, se puede avanzar a la siguiente pantalla y volver a la anterior en cualquiera de las pantallas mencionadas, excepto volver a la pantalla de *Splash.* Se puede acceder a la pantalla de preferencias desde las pantallas de selección de época y menú de edición.

Como se puede observar también, algunas pantallas tienen encima algún icono referente al servidor o a la base de datos local del dispositivo, esto significa que la pantalla que contiene el icono del servidor, accede al servidor, mientras que la pantalla que contiene el icono de la base de datos, accede a la base de datos local del dispositivo.

Este es un breve resumen del contenido de las pantallas mencionadas:

- **Splash**: Simplemente contiene el nombre de la aplicación y sirve como presentación del juego.
- **Pantalla inicial**: Contiene un botón para comenzar a jugar y el nombre de la aplicación.
- **Selección de época**: Accede al servidor para buscar todas las épocas posibles y las muestra para que puedan ser seleccionadas por el usuario.
- **Menú de edición**: Divido por 3 paneles (Panel de materiales de la época, panel de creación del bloque y panel de los bloques creados) con distintas funcionalidades. Se accede al servidor para recuperar las texturas utilizadas en la época seleccionada y recuperar las imágenes de las texturas. Por otro lado, se utiliza la base de datos local para guardar todos los bloques creados por el usuario en el modo edición.
- **Juego**: Pantalla utilizada para crear la construcción con los bloques creados por el usuario. Está divida en dos partes: el panel de los bloques creados y el tablero donde construir. Se accede al servidor para recuperar las imágenes de los bloques creados y se accede a la base de datos local para recuperar los bloques creados por el usuario en el modo edición.
- **Preferencias**: Contiene distintas opciones que pueden ser utilizadas por el usuario si lo desea. También contiene información del usuario que está utilizando la aplicación.

4.3 Diseño de la estructura de datos

Se han utilizado dos bases de datos diferentes para almacenar toda la información necesaria en la aplicación: una alojada en un servidor externo y otra como base de datos interna del dispositivo móvil. En los siguientes puntos se detallará la estructura utilizada en cada uno de los casos.

4.3.1 Diseño de la base de datos del servidor

Como ya se ha comentado anteriormente, se ha utilizado una base de datos en un servidor externo para poder almacenar todos los datos referentes a las épocas y texturas disponibles. Este servidor utiliza bases de datos PostgreSQL para almacenar sus datos.

Al tratarse de una base de datos pequeña, el modelo entidad-relación utilizado, que se puede observar en la *Ilustración 8*, también es pequeño como es lógico.

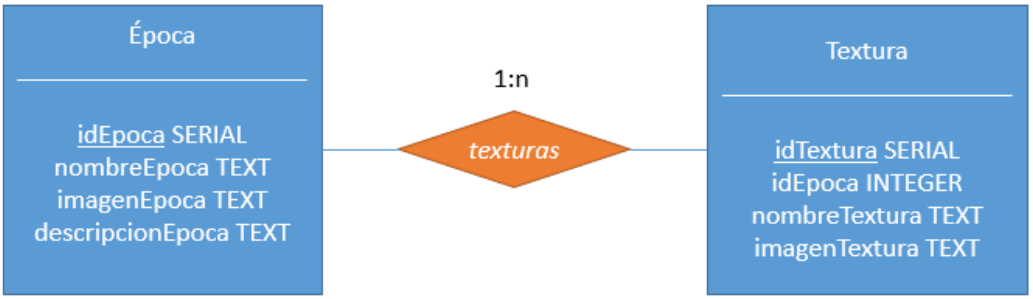

Ilustración 8. Entidad-Relación de la base de datos del servidor

Se puede observar cómo se han creado dos tablas: una para las **épocas** y otra para las **texturas**. La relación que hay entre ellas es 1:n, es decir, que las épocas pueden tener una o más texturas. Cabe destacar que los atributos subrayados son las claves primarias de cada una de las tablas.

La tabla de épocas contiene un id, que va incrementando automáticamente a medida que añades épocas, un nombre para la época y una descripción de la misma. También se puede apreciar que tiene una imagen de la época, pero en formato de cadena de caracteres. Esto significa que la imagen viene dada por una dirección HTTP que se refiere a dicha imagen, y con esta dirección obtienes la imagen mediante código.

La tabla de texturas contiene prácticamente la misma información que las épocas. Tiene un id, que también va incrementando automáticamente, un nombre y una imagen dada por una dirección HTTP. Sin embargo, contiene también el id de la época en la que está.

Después de ver las tablas creadas, se puede llegar a la conclusión de que se podría haber creado una tabla de época, otra de texturas y una que relacionase ambas con sus respectivos ids. No se escogió esta opción dado que el modelo fue desarrollado mientras se implementaba ya parte del código, y en ese momento se consideró más simple el diseño de la *Ilustración 8*. Sin embargo, no se descarta que en un futuro mantenimiento, se adapte al nuevo modelo descrito.

Para crear esta base de datos se ha utilizado un *script*, que se mostrará en el siguiente capítulo, para que pueda ser utilizado en todo momento por cualquier persona que tenga acceso al servidor.

4.3.2 Diseño de la base de datos del dispositivo móvil

Por otro lado, tenemos la base de datos interna del dispositivo móvil. Al igual que la base de datos del servidor, esta base de datos es muy pequeña, tan pequeña que solo contiene una tabla: la tabla de los bloques creados por el usuario en el modo edición.

Android utiliza SQLite como motor de base de datos. Ofrece características tan interesantes como su tamaño pequeño, ser transaccional y que no necesita un servidor [6].

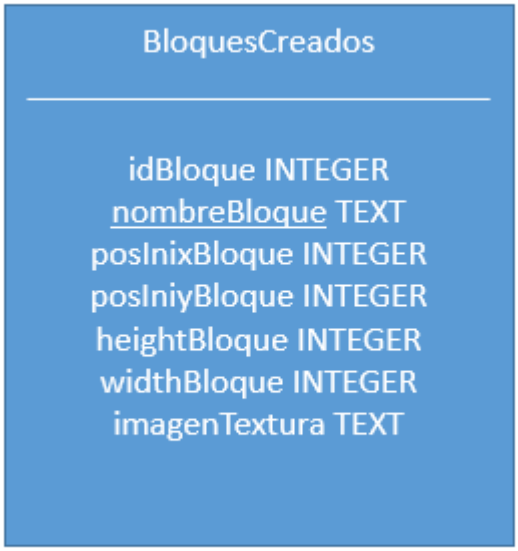

Ilustración 9. Tabla de bloques creados en la base de datos interna

En la *Ilustración 9* se puede ver la tabla donde se almacenará la información correspondiente a los bloques creados por el usuario, donde podemos ver que tiene como atributos un id, un nombre, la posición x e y del bloque, la altura, la anchura y por último, la ruta donde estará la imagen correspondiente al bloque.

El nombre es la clave primaria de esta tabla, porque esta tabla fue creada mientras se codificaba y en ese momento era necesario que los ids no fuesen únicos para realizar pruebas más rápidamente. De todos modos, en un futuro se pretende cambiar a clave primaria el id, para seguir con la misma idea que las otras tablas de la base de datos del servidor.

Para que el nombre del bloque creado sea único se ha creado una cadena que sigue este estilo:

Bloq + *TipoTextura* + *idBloque*

Con este estilo puedes obtener cadenas del tipo, “BloqAdobe1” o “BloqPiedra20”, por ejemplo.

Se ha utilizado la base de datos interna dado que no tiene sentido enviar más peticiones al servidor que pueden congestionar al mismo, ya que esta tabla se reiniciará cada vez que el usuario acceda a la pantalla del modo edición de bloques (no se reiniciará si el usuario vuelve de la pantalla de juego). Mientras que la otra base de datos se creó con un *script*, la base de datos interna ha sido creada mediante código para poder reiniciarla cuando sea necesario.

5. IMPLEMENTACIÓN

En este quinto capítulo se abordará todo lo relacionado con la implementación del proyecto. En esta parte se da forma a todos los requisitos que se habían pensado anteriormente en el *Capítulo 3*.

Es evidente que esta parte es una de las más importantes del proyecto, por no decir la que más, dado que aquí se creará toda la funcionalidad por la que se regirá el proyecto. Además, en el caso de que se den fallos cuando se realicen pruebas o se mantenga la aplicación, habrá que modificar la funcionalidad creada en ese punto.

A continuación, se explicarán todos los módulos creados en el proyecto, métodos utilizados y toda la información necesaria para obtener una visión global acerca de la funcionalidad de la aplicación.

Dividiremos en dos partes el capítulo: la parte cliente y la parte servidor, ya que son las dos bases del modelo cliente-servidor.

5.1 Cliente

La parte del cliente la constituye la aplicación Android. Aquí es donde se desarrolla toda la funcionalidad que va a llevar el juego.

Como hay bastante funcionalidad, se ha dividido el proyecto en varios módulos, acordes a su temática. También se dará una visión general de la interfaz desarrollada, con la que interactuará el usuario.

5.1.1 Módulos

A continuación, se comentarán los módulos utilizados en la aplicación y se detallará el objetivo de cada uno.

5.1.1.1 Módulo de la base de datos interna

En este módulo se configura y se realizan todas las acciones referentes a la base de datos local interna del dispositivo.

El módulo está formado por una única clase llamada **DatabaseAdapter.java**, donde está toda la funcionalidad. Dentro de esta clase se define la clase privada *DatabaseHelper* que es una subclase de *SQLiteOpenHelper*. El objetivo de esta clase es crear la base de datos y actualizarla cuando sea necesario.

Cuando se carga por primera vez, se llama al método encargado de crear la tabla y ejecutar la sentencia como podemos ver en la *Ilustración 10*. La sentencia no es más que una sentencia SQL de creación de una tabla.

```
//Creacion de la tabla de bloquesCreados
String str4 = "CREATE TABLE " + TABLE_BLOQUESCREADOS + " (" + ID_BLOQUE
        + " INTEGER, " + NOMBRE_BLOQUE
        + " TEXT PRIMARY KEY, " + POSINIX_BLOQUE + " INTEGER, " + POSINIY_BLOQUE + " INTEGER, "
        + HEIGHT_BLOQUE + " INTEGER, " + WIDTH_BLOQUE + " INTEGER, " + IMAGEN_TEXTURA + "TEXT);";

try {
    db.execSQL(str4);
} catch (SQLException e) {
    e.printStackTrace();
}
```

Ilustración 10. Código de la tabla de bloques creados

El constructor de la clase *DatabaseHelper* invoca al de la superclase pasando el nombre de la base de datos (DATABASE_NAME) y la versión (DATABASE_VERSION).

```
private static final String DATABASE_NAME = "ccc.db";
private static final int DATABASE_VERSION = 1;


private DatabaseHelper helper;
private SQLiteDatabase db;

public DatabaseAdapter(Context context) {
    helper = new DatabaseHelper(context);
}

private static class DatabaseHelper extends SQLiteOpenHelper {

    public DatabaseHelper(Context context) {
        super(context, DATABASE_NAME, null, DATABASE_VERSION);
    }

    public void onCreate(SQLiteDatabase db) {
        createTable(db);
    }
```

Ilustración 11. Constructor del DatabaseHelper

Esta clase también tiene un método *onUpgrade* que se invoca automáticamente cuando DATABASE_VERSION no coincide con la versión guardada en la base de datos. Como consecuencia, se borra la tabla y se crea de nuevo como se ve en la *Ilustración 12*.

```
public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
    db.execSQL("DROP TABLE IF EXISTS " + TABLE_BLOQUESCREADOS);
    createTable(db);
}
```

Ilustración 12. Método onUpgrade

Además hay un método *open* y otro *close*, que sirven para abrir y cerrar la base de datos cuando se necesite. Estos métodos serán llamados desde otras clases para poder acceder y cerrar la base de datos. El método *open* invoca al método *getWritableDatabase* que devuelve una referencia de tipo SQLiteDatabase para modificar los datos.

```
public DatabaseAdapter open() throws SQLException {
    db = helper.getWritableDatabase();
    return this;
}

public void close() {
    db.close();
}
```

Ilustración 13. Métodos open y close

Por último, contiene dos métodos más para añadir datos y recuperarlos. Para añadir los datos basta con utilizar un *insert* con todos los parámetros del bloque creado y para recuperarlos se utiliza el método *query*, con el que se realiza una consulta que recupere todos los bloques creados.

```
public long insertBloqueCreado(int id_bloque, String nombre_bloque,
        int posinix_bloque, int posiniy_bloque, int width_bloque,
        int height_bloque, String ruta) {

    ContentValues values = new ContentValues();
    values.put(ID_BLOQUE, id_bloque);
    values.put(NOMBRE_BLOQUE, nombre_bloque);
    values.put(POSINIX_BLOQUE, posinix_bloque);
    values.put(POSINIY_BLOQUE, posiniy_bloque);
    values.put(WIDTH_BLOQUE, width_bloque);
    values.put(HEIGHT_BLOQUE, height_bloque);
    values.put(IMAGEN_TEXTURA, ruta);
    return db.insert(TABLE_BLOQUESCREADOS, null, values);
}

public ArrayList<BloqueCreado> devuelveBloquesCreados(){
    ArrayList<BloqueCreado> array= new ArrayList<BloqueCreado>();
    Cursor cursor =db.query(TABLE_BLOQUESCREADOS, new String[] { ID_BLOQUE, NOMBRE_BLOQUE,
            POSINIX_BLOQUE, POSINIY_BLOQUE, HEIGHT_BLOQUE, WIDTH_BLOQUE,
            IMAGEN_TEXTURA}, null, null, null, null, null);

    if (cursor.moveToFirst()) {
        do {
        int index = cursor.getColumnIndex(NOMBRE_BLOQUE);
        String nombre = cursor.getString(index);
        int index2 = cursor.getColumnIndex(ID_BLOQUE);
        int idbloque = cursor.getInt(index2);
        int index3 = cursor.getColumnIndex(POSINIX_BLOQUE);
        int posinix = cursor.getInt(index3);
        int index4 = cursor.getColumnIndex(POSINIY_BLOQUE);
        int posiniy = cursor.getInt(index4);
        int index5 = cursor.getColumnIndex(HEIGHT_BLOQUE);
        int height = cursor.getInt(index5);
        int index6 = cursor.getColumnIndex(WIDTH_BLOQUE);
        int width = cursor.getInt(index6);
        int index7 = cursor.getColumnIndex(IMAGEN_TEXTURA);
        String ruta = cursor.getString(index7);

        BloqueCreado bloq= new BloqueCreado(idbloque,nombre,posinix,posiniy,height,width);
        bloq.setRutaImagen(ruta);
        array.add(bloq);
        } while (cursor.moveToNext());
    }
    return array;
```

Ilustración 14. Métodos de inserción y devolución de bloques creados

5.1.1.2 Módulo de las clases relacionadas con la vista

Este módulo está compuesto por las clases **Inicial.java**, **MainActivity.java**, **SelecciónEpoca.java**, **Tablero.java** y **JuegoView.java**.

Estas clases se encargan de obtener los elementos utilizados en la vista para poder interpretarlos y modificarlos según la necesidad del usuario. Como se puede observar hay cinco clases, una por cada XML descrito en el *Capítulo 4*.

Tanto Inicial.java como MainActivity.java se encargan de pasar de una pantalla a otra (Inicial a MainActivity, MainActivity a SelecciónEpoca).

5.1.1.2.1 SelecciónEpoca.java

SelecciónEpoca.java se encarga de obtener toda la información del servidor acerca de las épocas creadas para poder mostrarlas en la pantalla y poder interactuar con ellas. Para ello primero se conecta con la base de datos del servidor con el nombre de usuario y contraseña (se han tapado el usuario y la contraseña en la *Ilustración 15* por cuestión de privacidad). También se comprueba que esté en el proyecto el driver para poder interactuar con *postgreSQL*.

```
public boolean conectaBD(){
    StrictMode.ThreadPolicy policy = new StrictMode.ThreadPolicy.Builder().permitAll().build
    StrictMode.setThreadPolicy(policy);

    try {
        Class.forName("org.postgresql.Driver");
    } catch (ClassNotFoundException e) {
        System.out.println("No driver.");
        e.printStackTrace();
        return false;
    }

    try {
        connection = DriverManager.getConnection(
                "jdbc:postgresql://sacha.ii.uam.es:5432/sergioTFG",
                [redacted],[redacted]);
    } catch (SQLException e) {
        System.out.println("Conexion fallada.");
        e.printStackTrace();
        return false;

    }
```

Ilustración 15. Conexión a la base de datos del servidor

Una vez que estamos conectados a la base de datos, se realiza una consulta donde recuperar todas las épocas.

```
if(conectaBD()){
    ResultSet resultados = consultar("SELECT * FROM epocas");
    if (resultados != null) {
        try {
            while (resultados.next()) {
                pinta(resultados.getInt("idepoca")
                        ,resultados.getString("nombreepoca")
                        ,resultados.getString("imagenepoca"),
                        resultados.getString("descripcionepoca"));
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}else{
    Toast.makeText(this, "No se ha podido conectar al servidor",
            Toast.LENGTH_SHORT).show();
}
```

Ilustración 16. Consulta de épocas

Con los resultados obtenidos, creamos unas imágenes con las fotos representativas de las épocas. Además, si el usuario pulsa la imagen, saltará un dialogo con la descripción de la misma. Estas imágenes se han creado con el elemento *ImageButton*, que permite tener un controlador como si se tratase un botón pero que al mismo tiempo muestra una imagen.

Como se puede observar en la *Ilustración 17*, *DownloadFilesTask* es una tarea asíncrona que se encarga de actualizar la imagen en la foto (las tareas asíncronas se verán con más detalle en otro módulo). De esta manera, evita que el usuario tenga que esperar a que se carguen todas las fotos.

```
public void pinta(final int idEpoca,final String nombre, String ruta, final String des) {
    TextView nombreText = new TextView(this);
    nombreText.setText(nombre);
    nombreText.setGravity(Gravity.CENTER);
    int color = getResources().getColor(R.color.blanco);
    nombreText.setTextColor(color);
    linearLayout.addView(nombreText);

    ImageButton imagen = new ImageButton(this);
    imagen.setAdjustViewBounds(true);
    new DownloadFilesTask().execute(imagen,ruta);
    imagen.setOnClickListener(new View.OnClickListener() {
        public void onClick(View view) {
            dialog(idEpoca,nombre, des);

        }
    });
    linearLayout.addView(imagen);
}
```

Ilustración 17. Código para mostrar las imágenes de las épocas

5.1.1.2.2 Tablero.java

Tablero.java y JuegoView.java son las clases fundamentales en esta aplicación, ya que ambas se encargan de realizar los objetivos principales del juego. Tablero.java se encarga de configurar todo el tablero del modo edición. Como ya se ha comentado en anteriores puntos, este modo edición está compuesto de las siguientes partes:

- Parte de las texturas que tiene la época seleccionada
- Parte de configuración del bloque
- Parte de los bloques creados

El primer paso que sigue esta actividad es reiniciar la base de datos para eliminar los antiguos bloques creados por la última sesión ejecutada por el usuario. Posteriormente, se realiza una consulta en la base de datos, con el id de la época seleccionada en la pantalla anterior, para obtener las texturas disponibles para dicha época. Con los resultados obtenidos de la consulta, se realiza el mismo procedimiento que en SelecciónEpoca.java para pintar las texturas en el panel de los materiales (también se utiliza una tarea asíncrona).

Una vez que se ha creado el panel de los materiales, se crea el panel de edición de bloque que consta a su vez de tres subpaneles:

- Panel del bloque por defecto
- Panel de los botones de acciones
- Panel de dimensiones del bloque

Cada uno de estos paneles es un linearLayout (panel que dispone todos los elementos por filas) con distintos ids, que se utilizarán para poder acceder a ellos cuando sea necesario.

Para el primer panel, se crea un bloque por defecto que interpretará todos los cambios que vaya realizando el usuario. Este bloque por defecto será del primer material disponible en la época seleccionada.

El usuario tiene la opción de cortar el bloque manualmente con el dedo (si tiene activada la preferencia que se lo permita) con una línea vertical de abajo a arriba. El corte manual es muy exacto, por lo que a veces resulta complicado realizar un corte (esta circunstancia pretende solventarse en un futuro mantenimiento).

En la *Ilustración 18* se puede observar el algoritmo utilizado para el corte manual. Hay dos *arrays* de posiciones (x e y) que se van recorriendo en bucle para comprobar si algún punto está situado dentro del bloque. En el momento que se detecta el punto dentro del bloque, se traza una línea vertical en ese punto y se calculan las anchuras que hay entre los lados y la línea trazada. Finalmente se queda con la anchura más grande de las dos obtenidas y cambia la anchura total del bloque por la elegida.

```
while(i<posx.size()){
    if((posx.get(i)>=bloque.getPosinix())&&
            (posx.get(i)<(bloque.getPosinix()+bloque.getWidthBloq()))
            &&(posy.get(i)>=bloque.getPosiniy())&&
            (posy.get(i)<(bloque.getPosiniy()+bloque.getHeightBloq()))){
        posxpartida= this.posx.get(i).intValue();
        widthpartidaderecha=(bloque.getPosinix()+bloque.getWidthBloq())-posxpartida;
        widthpartidaizquierda=bloque.getWidthBloq()-widthpartidaderecha;
        if(widthpartidaderecha>=widthpartidaizquierda){
            bloque.setWidthBloq(widthpartidaderecha);
            editWidth.setText(Integer.toString(widthpartidaderecha));
            Toast.makeText(v.getContext(), "Bloque partido (Parte derecha)",
                    Toast.LENGTH_SHORT).show();
        }else{
            bloque.setWidthBloq(widthpartidaizquierda);
            editWidth.setText(Integer.toString(widthpartidaizquierda));
            Toast.makeText(v.getContext(), "Bloque partido (Parte izquierda)",
                    Toast.LENGTH_SHORT).show();
        }
        bloque.flag=1;
        bloque.invalidate();
        break;
    }
    i++;
}
posx.clear();
posy.clear();
flag=false; }
```

Ilustración 18. Código del corte manual

Una vez terminada esta parte, pasamos a la parte de los botones donde el usuario podrá realizar distintas acciones sobre el bloque. A continuación se presentan las acciones que puede tomar el usuario:

- **Deshacer cambios**: Basta con cambiar los valores actuales por los valores iniciales (que han sido guardados en variables).
- **Rotar**: Simplemente se intercambian la altura por la anchura y viceversa.
- **Crear bloque**: Se crea un bloque con los valores actuales configurados por el usuario y se inserta en la base de datos local.
- **Continuar**: Simplemente pasa al tablero de juego. Antes comprueba los bloques creados, ya que si no hay ningún bloque creado se crean 25 copias con altura y anchura de 100 pixeles. La textura de los bloques por defecto será la primera textura disponible en la época seleccionada.

El panel de dimensiones de bloque contiene la información del tamaño del bloque y el nombre del material compuesto. Las medidas del bloque (en pixeles) pueden ser editadas por el usuario y finalmente cambiadas pulsando el botón "*Cambiar*". Estas medidas pueden ser como máximo de 180 pixeles (es una medida aproximada en relación a la proporcionalidad de una pantalla de siete pulgadas) tanto para altura como para anchura.

Por último, queda el panel de los bloques creados en donde se irán añadiendo todos los bloques que vaya creando el usuario con "*Crear bloque*".

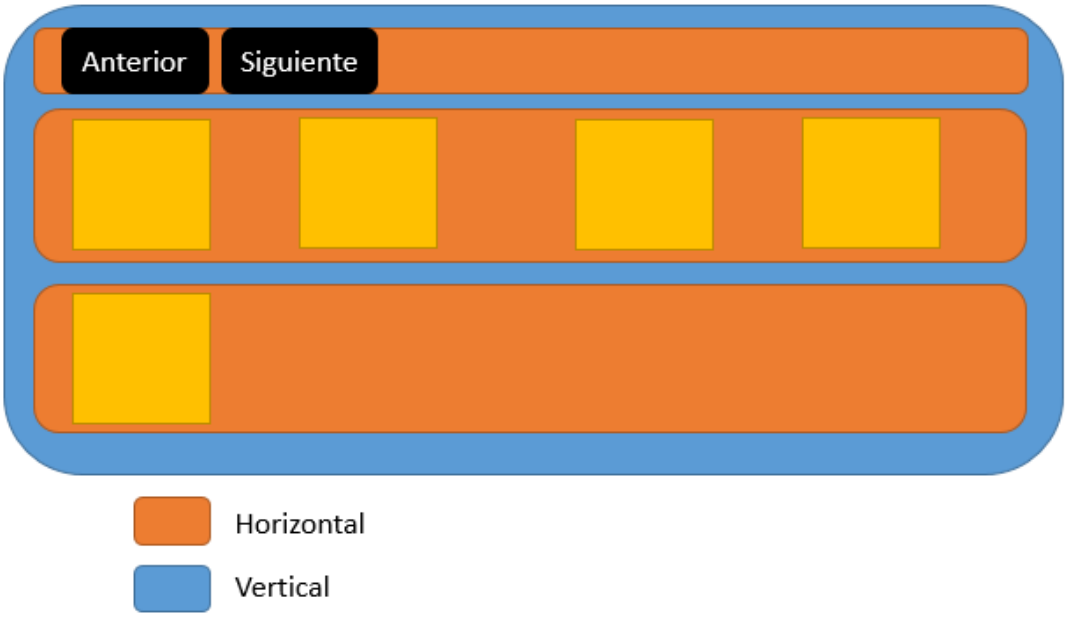

Ilustración 19. Layouts utilizados en el panel de los bloques creados

En la *Ilustración 19* se pueden observar los *layouts* utilizados para este panel, así como la orientación de cada uno. El *layout* principal (orientación vertical) contiene tres *layout* horizontales: uno para los botones de "*Anterior*" o "*Siguiente*" y otros dos para los bloques creados. Esta configuración permite un máximo de ocho bloques para mostrar. Si se quiere observar los demás bloques creados, se debe utilizar la combinación de botones del primer *layout*. Estos botones solo aparecen cuando realmente hay la posibilidad de ver algún bloque anterior o posterior. Esto provoca que cada vez que se utilice un botón, haya que reconstruir el tablero desde cero.

Se utiliza una variable para saber cuáles se están mostrando. Esta variable se actualiza cada vez que se completan los ocho espacios disponibles y se acumula a los bloques creados anteriormente. Es decir, si muestro seis bloques y previamente tengo creados otros ocho, que no se están mostrando, el valor de la variable es de ocho, dado que todavía hay algunos huecos disponibles. Con esta variable creo las condiciones necesarias para que aparezcan los botones "*Anterior*" y "*Siguiente*".

```
//Significa que no viene de la 2a pantalla
if(ultimoIdBloque!=8){
    Button boton = new Button(this);
    boton.setText("Anterior");
    boton.setBackgroundResource(R.drawable.buttonshape);
    isAnterior=true;
    boton.setOnClickListener(new View.OnClickListener() {
        public void onClick(View view) {
            tablero.removeAllViewsInLayout();
            ultimoIdBloque=ultimoIdBloque-8;
            reconstruirTablero();
        }
    });
    linearlayoutInterno.addView(boton);
}
```

```
//Condición para que aparezca siguiente
if(bloquesCreados.size()>=ultimoIdBloque){
    Button boton2 = new Button(this);
    boton2.setText("Siguiente");
    boton2.setBackgroundResource(R.drawable.buttonshape);
    isSiguiente=true;
    boton2.setOnClickListener(new View.OnClickListener() {
        public void onClick(View view) {
            tablero.removeAllViewsInLayout();
            ultimoIdBloque=ultimoIdBloque+8;
            reconstruirTablero();
        }
    });
    linearlayoutInterno.addView(boton2);
}
```

Ilustración 20. Condiciones para que aparezcan los botones Anterior y Siguiente

Para pintar los bloques, según la situación en el array, se ha utilizado el bucle que se puede observar en la *Ilustración 21*.

```
for(int i=(ultimoIdBloque-8);i<ultimoIdBloque;i++){
    if(i>=bloquesCreados.size()){break; }
    if(contBloqTotal==8){break; }
    if(contBloq==4){
        linearlayoutInterno = new LinearLayout(this);
        linearlayoutInterno.setOrientation(LinearLayout.HORIZONTAL);
        linearlayoutInterno.setLayoutParams(new LayoutParams
                (LayoutParams.WRAP_CONTENT,LayoutParams.WRAP_CONTENT));
        linearlayoutInterno.setTag("linear1");
        linearlayoutInterno.setOnTouchListener(this);
        tablero.addView(linearlayoutInterno);
        contBloq=0;
    }
    Bloque bloqAdd= new Bloque(this,bloquesCreados.get(i).getId(),
            bloquesCreados.get(i).getNombre(),bloquesCreados.get(i).getPosinix(),
            bloquesCreados.get(i).getPosiniy(),bloquesCreados.get(i).getHeightBloq(),
            bloquesCreados.get(i).getWidthBloq());
    bloqAdd.setImagen(bloquesCreados.get(i).getImagen());
    LinearLayout linearlayout = new LinearLayout(this);
    linearlayout.setOrientation(LinearLayout.VERTICAL);
    linearlayout.setLayoutParams(new LayoutParams(180,180));
    linearlayout.setPadding(10, 10, 10, 10);
    linearlayout.addView(bloqAdd);
    linearlayout.setTag("bloque"+contadorBloquesTotal);
    linearlayout.setOnTouchListener(this);
    linearlayoutInterno.addView(linearlayout);
    contBloq++;
    contBloqTotal++;
}
```

Ilustración 21. Bucle de pintado de los bloques apropiados

5.1.1.2.3 JuegoView.java

Por otra parte está la clase JuegoView.java, donde se realiza la funcionalidad necesaria para que los bloques interactúen y colisionen entre ellos para poder construir. En esta clase están los algoritmos empleados para provocar la sensación de física realista en el usuario.

JuegoView.java no es una actividad como tal, es una *SurfaceView*. Este elemento proporciona una superficie de dibujo. Puedes controlar el formato de esta superficie ya que *SurfaceView* se encargará de colocar la superficie en el lugar correcto de la pantalla. De esta manera, hay una clase que si es la actividad (**Juego.java**) y crea la *SurfaceView*.

Juego.java se encarga de pintar todos los bloques creados por el usuario, que han sido recuperados desde la base de datos local, y del método de selección de los mismos. El usuario selecciona un bloque, el sistema marca el bloque como seleccionado con un color amarillo, y se pasa dicha selección al *SurfaceView* para que el bloque caiga cuando el usuario pulse la posición donde va a caer. A continuación, se volverá de color rojo el bloque seleccionado en el panel, para evitar que el usuario pueda volver a elegirlo. Esta selección se puede ver de manera más ilustrada en el Anexo B.

Volviendo a **JuegoView.java**, esta clase se encarga de realizar la simulación física por la que se regirá cada uno de los bloques. Cuando se ejecuta por primera vez, se crea un hilo que ejecuta tres acciones en este orden:

1. Limpiar pantalla de bloques
2. Mover bloque
3. Pintar todos los bloques.

De esta manera, cada vez que se inserta un bloque en la pantalla (y hasta que el bloque se quede quieto), se limpia la pantalla, se ejecuta el siguiente movimiento que realizará el bloque y se pinta con su nueva posición (también se pintan los bloques restantes, ya que la pantalla ha sido limpiada previamente).

Para simular la gravedad mientras cae el bloque, hay una función externa que depende del tiempo. En la implementación actual se usa una versión de caída libre sin rozamiento con el aire. En el caso en que el bloque que está cayendo sufra algún tipo de colisión con el suelo u otros bloques, el programa ejecutará una serie de algoritmos basados en cálculos geométricos que simulan un movimiento de rotación, de parada o de colisión múltiple cuando sea necesario. Tanto en la simulación de la gravedad como en los algoritmos de colisión, se han estructurado y modulado de tal manera que se puedan variar las fórmulas o datos concretos sin que se tengan que realizar cambios significativos.

Las leyes físicas básicas de la dinámica, especialmente las relacionadas con colisiones, se han implementado en la aplicación de una forma simplificada, donde se ha insistido especialmente en la caída libre de un bloque y sus posibles interacciones con los bloques ya presentes en el espacio. Estos bloques se han considerado inmóviles en

primera aproximación, siendo el bloque que está cayendo el único con posibilidad de movimiento.

El bloque irá realizando distintos movimientos mientras está cayendo hasta que llegue el momento de que pare. Para comprobar cuál es el siguiente movimiento que tiene que realizar el bloque, se ha creado un algoritmo que contempla los casos por los que puede pasar el bloque y calcula el siguiente movimiento que le toca ejecutar. En la *Ilustración 22* se puede ver puede ver el pseudocódigo seguido para este método.

```
bq= bloque que esta descendiendo

posActualizada = posición inicial y del bloque + gravedad
Sino has llegado al final de la pantalla
    Bucle i=bloque0 hasta bloqueSize
        si vaAChocar bq con i
            sino vaARotar bq con i
                Actualizo posY del bloque con posActualizada (Sigue descendiendo)
            si vaARotar bq con i
                Actualizo posY del bloque con posActualizada

                si ningún otro bloque me impide la rotación
                    me quedo con el id del bloque contra el que rota (Bloque va a rotar)
                sino el bloque colisiona con otro bloque lateral

    Fin bucle
Si has llegado al final de la pantalla
    Actualizo posY del bloque con posActualizada (Bloque colisiona contra el final de la pantalla)
```

Ilustración 22. Método move 1ª parte

En primer lugar se calcula la siguiente posición en el eje y, que tendría que tomar el bloque que desciende sino se encuentra con ningún obstáculo. Se comprueba si ha llegado al final de la pantalla, si no ha llegado pasamos a comprobar una serie de acciones con todos los bloques restantes que hay en el lienzo. Primero comprueba si va a chocar con algún bloque, si efectivamente va a chocar hay que comprobar si el bloque se sostendrá correctamente encima del bloque con el que va a chocar, o si por el contrario no se sostendrá y provocará una rotación para un lado.

Sabiendo que el bloque va a rotar, hay que comprobar que ningún otro bloque del lienzo impida esa rotación (el bloque se sujeta por el otro lado con otro bloque como se puede ver en la *Ilustración 23*).

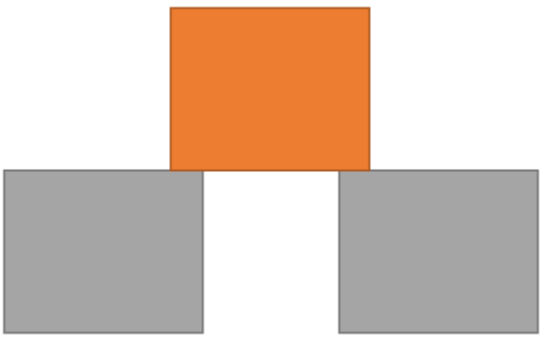

Ilustración 23. El bloque no puede rotar

Esta es la base que seguirán los bloques para que se muevan por el tablero. Ahora se va a profundizar en los métodos que se encargan de las distintas acciones que se comprueban en el algoritmo principal.

Bloque llega al final de la pantalla

Para empezar comprobaremos como se **detecta si el bloque ha llegado al final de la pantalla**. Este algoritmo es muy simple dado que sumamos la altura del bloque con la siguiente posición que tomaría el bloque en el eje y. Si el resultado es mayor o igual que la altura de la pantalla, significa que el bloque ha llegado al final de la pantalla, por lo que el bloque se para.

Bloque colisiona con otro bloque

Ahora vamos a explicar el algoritmo que **detecta si el bloque que desciende colisionará con otro bloque**. Estando en el bucle que recorre los demás bloques, se comprueba si la siguiente posición en el eje y del bloque que cae es mayor que la posición y del bloque con el que está comparando, significa que va a colisionar. Aunque puede que no colisione, ya que el bloque podría caer por el lateral del bloque con el que comparas. En la *Ilustración 24* se puede observar la comprobación de los laterales del bloque en el eje x.

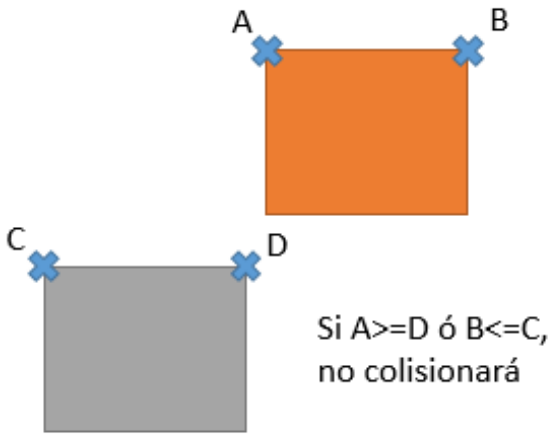

Ilustración 24. Comparación lateral del bloque

Bloque rota

Pasamos a comprobar ahora **si se detecta que el bloque va a rotar**. El algoritmo implementado para controlar esta acción, primero calcula el punto medio del bloque que está descendiendo y compara este punto medio con las esquinas izquierda y derecha, ya que si el punto medio es mayor que la esquina derecha significa que rotará, al igual que si el punto medio es menor que la esquina izquierda rotará también. Esto se puede resumir en la *Ilustración 25*.

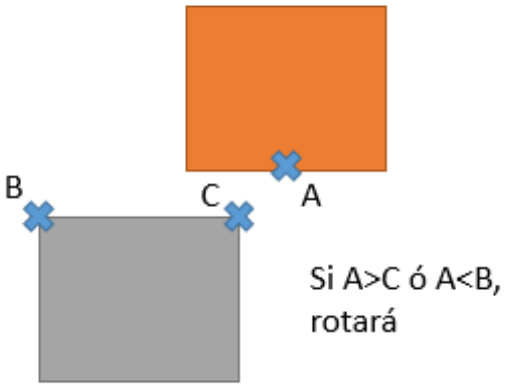

Ilustración 25. Comprobar si el bloque rotará

Sin embargo, puede que al intentar rotar, el bloque no pueda porque tiene otro bloque que se lo impide como en la *Ilustración 23*. Para ello se comprueban que todos los bloques de la derecha (si rota hacia la derecha) o los bloques de la izquierda (si rota hacia la izquierda) no interfieran en la rotación.

Volviendo al tema de la rotación, se ha implementado un algoritmo que **simula la rotación de un bloque** a través de cálculos geométricos.

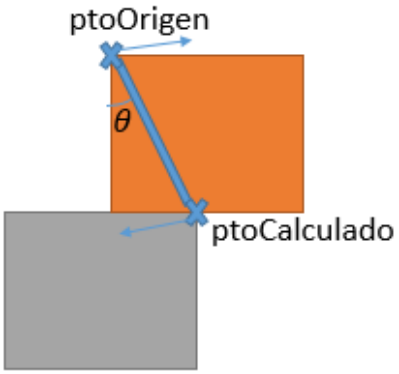

Ilustración 26. Dibujo geométrico para la rotación

Los cálculos geométricos para una rotación derecha del bloque son los siguientes:

1. Se calcula r=√ (a²+b²) y θ = arctg(b/a) para pasar el punto que rota a coordenadas polares.
2. Actualizamos zeta con $\theta = \theta$ - ang, donde ang es el incremento de ángulo que hemos impuesto en la aplicación (En nuestro caso es de 10º, pero puede variarse si así se desea).
3. Volvemos a pasar a coordenadas cartesianas con a'= rcos θ y b'=rsen θ.
4. Pasamos de nuevo el punto (a',b') a cartesianas y restamos el desplazamiento para mantener el punto en la misma posición.

Estos cálculos se realizan hasta que el bloque ha girado 90º (o si el bloque ha colisionado con otro). Cuando ha girado esos 90º, el bloque sigue descendiendo hasta que se encuentre el final de la pantalla u otro bloque que le obstaculice.

Este cálculo geométrico se puede observar en la *Ilustración 27*.

```
public Point rotacion(BloqueCreado bq, BloqueCreado bqQuieto, int angRotacion, int ladoRotacion){
    int x1=bq.getPosinix();
    int y1=bq.getPosiniy();
    int x2=bq.getPosinix();
    int x3;
    if(ladoRotacion==0){
        x3=bqQuieto.getPosinix()+bqQuieto.getWidthBloq();
    }else{
        x3=bqQuieto.getPosinix();
    }
    double r=Math.sqrt((double)(Math.pow((x3-x2),2)+Math.pow(bq.getHeightBloq(),2)));
    double ang=Math.atan(((double)(x3-x2))/((double)(bq.getHeightBloq())));
    //Pasar de radianes a grados
    double angGrado=Math.toDegrees(ang);
    angGrado=angGrado-angRotacion;
    //Pasar de grados a radianes
    double angRadianes=Math.toRadians(angGrado);
    //Volvemos a pasar a coord.cartesianas
    double ptoEsqDerechaXNuevo=(r*(Math.cos(angRadianes)));
    double ptoEsqDerechaYNuevo=(r*(Math.sin(angRadianes)));
    Point p= new Point((int)(x1+((x3-x2)-ptoEsqDerechaYNuevo)),
            (int)(y1+((bq.getHeightBloq()-ptoEsqDerechaXNuevo))));
    return p;
```

Ilustración 27. Código de la rotación

El método que contiene el algoritmo principal (cuyo pseudocódigo es el de la *Ilustración 22*) tiene una segunda parte encargada de ir realizando la rotación en cada paso. El pseudocódigo de esta parte se puede ver en la *Ilustración 28*.

```
bq= bloque que esta descendiendo

si bq.Rotando==true
    si el angRotacion del bloque es distinto de 90 y es derecha
    o que el angRotacion del bloque es distinto de -90 y es izquierda
        si colisionara contra otro bloque mientras rota
            paro de rotar
        sino
            si es rotacion derecha
                ang=ang+incrementoAng
            si es rotacion izquierda
                ang=ang-incrementoAng
            bloque rota con nuevo ang

    sino
        ang=0
        bq.Rotando=false
```

Ilustración 28. Método move 2ª parte

Esta segunda parte del algoritmo principal comprueba si el bloque está rotando, si es así comprueba que el ángulo no sea de 90º hacia la derecha o de -90º hacia la izquierda, ya que eso implicaría que el bloque está en el último paso de la rotación. Si no se da esta comparación, significa que el bloque está rotando, por lo que primero comprueba si va a colisionar con otro bloque y sino choca se aumenta el ángulo y se rota. Esta situación se puede observar en la *Ilustración 29*.

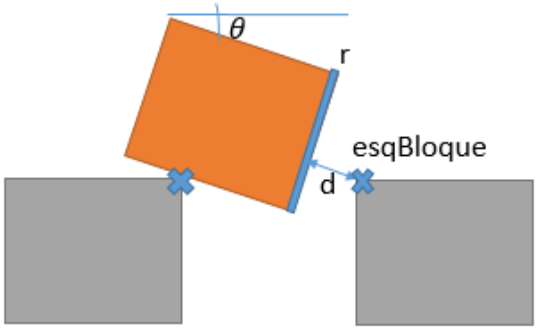

Ilustración 29. Dibujo geométrico para la colisión mientras rota

Bloque colisiona mientras rota

Para comprobar si el bloque va a colisionar con otro bloque, se ha establecido como condición de parada que la distancia *d* entre la recta *r* y el punto *esqBloque* sea negativa, o dicho de otro modo, que el punto *esqBloque* se encuentre en el interior del bloque. En la *Ilustración 30* se puede ver el código que comprueba este caso.

```
public boolean comprobacionSiChocaMientrasRota(BloqueCreado bq, int idBloqueRotacion,
        int resultadoRotacion, int angulo){
    double angRadianes=Math.toRadians(angulo);
    double x = 0,y = 0,pendiente = 0;
    if(resultadoRotacion==0){
        x=(bq.getWidthBloq()*(Math.cos(angRadianes)));
        y=(-1*(bq.getWidthBloq()*(Math.sin(angRadianes))));
        pendiente=(x/(-1*y));
    }else{
        x=(bq.getHeightBloq()*(Math.cos(angRadianes)));
        y=(-1*(bq.getHeightBloq()*(Math.sin(angRadianes))));
        pendiente=(x/(-1*y));
    }
    //Calculo ecuacion de la recta
    //y=ax+b
    double a=pendiente, b=y-(a*x), c, d, restaY, restaX, xPrima;

    for(BloqueCreado bqQuieto: listaBloquesTablero){
        if((bqQuieto.getIdBloque()!=idBloqueRotacion)&&(bq.getIdBloque()!=bqQuieto.getIdBloque())){
            //compruebo solo los bloques que esten a la derecha del que esta rotando
            if((bq.getPosinix()<bqQuieto.getPosinix())&&(resultadoRotacion==0)
                ||(bq.getPosinix()>bqQuieto.getPosinix())&&(resultadoRotacion==1)){
                //Calculo ecuacion recta perpendicular
                if(resultadoRotacion==0){
                    pendiente=(y/x);
                    c=pendiente;
                    restaY=bq.getPosiniy()-bqQuieto.getPosiniy();
                    restaX=bqQuieto.getPosinix()-bq.getPosinix();
                    d=restaY-(c*(restaX));
                    //Interseccion
                    xPrima=((d-b)/(a-c));
                    xPrima += bq.getPosinix();
                    if(bqQuieto.getPosinix()<=xPrima){//Choca
                        return true;
                    }
                }else{
                    pendiente=(y/x);
                    c=pendiente;
                    restaY=bq.getPosiniy()-bqQuieto.getPosiniy();
                    restaX=bqQuieto.getPosinix()-bq.getPosinix();
                    d=restaY-(c*(restaX));
                    //Interseccion
                    xPrima=((d-b)/(a-c));
                    xPrima += bq.getPosinix();
                    if(xPrima<=(bqQuieto.getPosinix()+bq.getWidthBloq())){//Choca
                        return true;
                    }
                }
            }
```

Ilustración 30. Código de comprobación de colisión mientras rota

También se ha creado un método para comprobar si los bloques están quietos. Este método es importante para que el hilo sepa parar y no realice iteraciones innecesarias.

```
public boolean bloquesQuietos(){
    for(BloqueCreado bq:listaBloquesTablero){
        if(!bq.isQuieto()){
            return false;
        }
    }
    return true;
}
```

Ilustración 31. Código para comprobar si los bloques están quietos

Hasta aquí se han explicado todos los algoritmos empleados en el tablero. Sin embargo, no se ha comentado como se pintan los bloques en el lienzo. La *Ilustración 32* muestra cómo se recorren todos los bloques que hay en el lienzo, y se pintan como un *Bitmap Escalado*. Si el bloque está rotando, se aplica el ángulo de rotación y luego se pinta.

```
public void pintaBloques(Canvas canvas){
    Paint paint = new Paint(Paint.FILTER_BITMAP_FLAG);

    for(BloqueCreado bq:listaBloquesTablero){
        Bitmap bitmap = bq.getBitmapBloque();
        Bitmap bitmapnew = Bitmap.createScaledBitmap(bitmap,bq.getWidthBloq(),
                bq.getHeightBloq(), false);

       if(!bq.isRotando()){
               canvas.drawBitmap(bitmapnew,bq.getPosinix(), bq.getPosiniy(), paint);
       }else{
           if(bq.getAngRotacion()<=90){
               Matrix rotator = new Matrix();
               int x = bq.getPosinix();
               int y = bq.getPosiniy();
               rotator.setTranslate(x, y);
               rotator.preRotate(bq.getAngRotacion());
               canvas.drawBitmap(bitmapnew, rotator, paint);
           }
       }
    }
}
```

Ilustración 32. Código para dibujar los bloques en el lienzo

Para terminar, cuando el usuario finaliza la partida se realiza una captura de pantalla de la construcción y se prepara un correo para enviar al profesor. Para realizar la captura de pantalla, se crea un *bitmap* con todo el contenido de la pantalla. Y para preparar el correo se utilizan los *Intents* de acceso a otra actividad. Todo esto se puede visualizar en la *Ilustración 33* e *ilustración 34*.

```
public Bitmap pantallazo(){
    Bitmap bit = Bitmap.createBitmap(width, height, Config.ARGB_8888);
    Canvas c = new Canvas(bit);
    c.drawColor(Color.WHITE);
    pintaBloques(c);
    return bit;
}
```

Ilustración 33. Código para realizar la captura de pantalla

```
public void enviaCorreoProfesor(){
    SharedPreferences sharedPreferences =
            PreferenceManager.getDefaultSharedPreferences(this);
    String nombre = null;
    String correo = null;
    if (sharedPreferences.contains(CCCPreference.NOMBRE_KEY)){
        nombre = sharedPreferences.getString(CCCPreference.NOMBRE_KEY,
                CCCPreference.NOMBRE_DEFAULT);
    }
    if (sharedPreferences.contains(CCCPreference.CORREO_KEY)){
        correo = sharedPreferences.getString(CCCPreference.CORREO_KEY,
                CCCPreference.CORREO_DEFAULT);
    }
    //Poner el nombre del alumno
    String mensaje="Soy "+nombre+" y te adjunto esta imagen con la construccion realizada.\n" +
            "Un saludo.\n\n\nEnviado desde Historium.";
    Intent emailIntent = new Intent(Intent.ACTION_SEND);
    emailIntent.setData(Uri.parse("mailto:"));
    //Poner correo de preferencias
    String[] to = {correo};
    emailIntent.putExtra(Intent.EXTRA_EMAIL, to);
    emailIntent.putExtra(Intent.EXTRA_SUBJECT, "Historium");
    emailIntent.putExtra(Intent.EXTRA_TEXT, mensaje);
    Uri uri = Uri.fromFile(new File(Environment.getExternalStorageDirectory(), "capture.png"));
    emailIntent.putExtra(Intent.EXTRA_STREAM, uri);
    emailIntent.setType("message/rfc822");
    startActivity(Intent.createChooser(emailIntent, "Email "));
```

Ilustración 34. Código para preparar el correo

5.1.1.3 Módulo de las tareas asíncronas

Este módulo se encarga de realizar peticiones HTTP al servidor externo, sin que se vea modificado el rendimiento. Para ello se utiliza la clase *AsyncTask* que contiene las funciones necesarias para realizar acciones antes y después de la petición según se requiera. En esta aplicación, se utiliza esta clase para obtener las épocas y texturas del servidor de manera que el usuario no tenga que esperar a que se carguen todas las épocas o texturas para poder ir utilizando la aplicación.

Para realizar esta labor se crea una clase que extiende *AsyncTask* y que contiene los siguientes métodos:

- **doInBackground**: realiza en segundo plano la tarea que se desee
- **onProgressUpdate**: se utiliza por si quieres indicar algo mientras se está actualizando.
- **onPostExecute**: realiza las acciones que desees una vez que haya terminado de realizar las tareas de segundo plano.

En la *Ilustración 35* se puede observar en el método *doInBackground* como recuperamos la imagen a través de la petición HTTP. Una vez que ha terminado el segundo plano, esta imagen se carga en el objeto final en el método *onPostExecute*.

```
private class DownloadFilesTask extends AsyncTask<Object, Integer, Bitmap> {
    ImageButton imagen;
     protected Bitmap doInBackground(Object... params) {
         imagen = (ImageButton) params[0];
         //Recuperamos la ruta de la imagen
         String path = (String) params[1];
         Bitmap loadedImage;
         URL imageUrl = null;
         try {
            imageUrl = new URL(path);
            HttpURLConnection conn = (HttpURLConnection) imageUrl
                    .openConnection();
            conn.connect();
            loadedImage = BitmapFactory.decodeStream(conn.getInputStream());
        } catch (IOException e) {
            e.printStackTrace();
            return null;
        }
         return loadedImage;
     }

     protected void onProgressUpdate(Integer...a){
         Log.e("Debug","You are in progress update ... " + a[0]);
     }

     protected void onPostExecute(Bitmap loadedImage) {
         imagen.setImageBitmap(loadedImage);
         Log.e("Debug","onPostExecute()");
     }
```

Ilustración 35. Código de tarea asíncrona

5.1.1.4 Módulo de las clases representativas

Este módulo contiene aquellas clases que representan los distintos elementos que contiene el juego. Estos elementos serian **Epoca.java**, **Textura.java**, **Bloque.java** y **BloqueCreado.java**.

La razón de que haya dos tipos de bloques se debe a que Bloque.java extiende de la clase *View*, mientras que en BloqueCreado.java no. Es necesario que Bloque.java extienda de esa clase para poder mostrarse de forma correcta en el modo edición. BloqueCreado.java no necesita de esa clase, ya que esta clase será utilizada en una *SurfaceView*.

En la *Ilustración 36* se pueden ver los atributos de cada una de las clases.

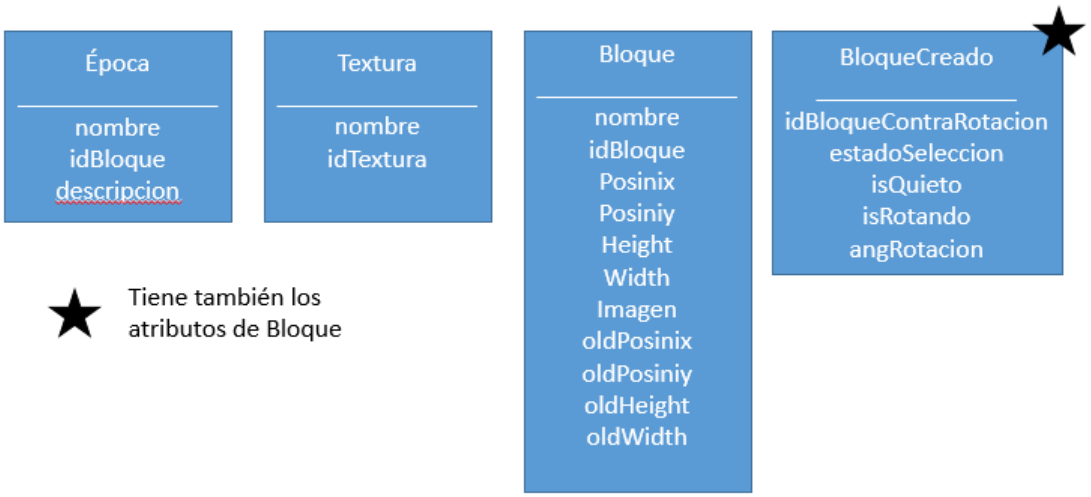

Ilustración 36. Atributos de las clases representativas

5.1.1.5 Módulo de preferencias

Se han creado las siguientes preferencias que puede activar el usuario si así lo desea:

- **Cambiar nombre del alumno**: En esta opción el usuario deberá introducir su nombre para que posteriormente sea utilizado en el correo.
- **Cambiar correo del profesor**: En esta opción el usuario deberá introducir el correo de su profesor para que posteriormente sea utilizado en el correo.
- **Música**: Si se activa, sonará música en el menú de edición.
- **Corte Manual**: Si se activa, el usuario podrá cortar el bloque manualmente.
- **Bloques infinitos**: Si se activa, los bloques creados por el usuario serán infinitos.

El usuario puede acceder a estas preferencias a través del botón menú, en las pantallas de selección de época y edición de bloques, si pulsa en la pestaña "*Opciones*".

Este menú tiene distintas configuraciones según la pantalla en la que se encuentre el usuario:

- **SelecciónEpoca.java**: Tiene la opción de acceder a las preferencias y de obtener información sobre la aplicación (Acerca De).
- **Tablero.java**: Las mismas opciones que SelecciónEpoca.java pero también tiene la posibilidad de reiniciar los bloques creados.
- **JuegoView.java**: Tiene la opción de reiniciar o finalizar la partida.

En la *Ilustración 37* se puede ver como se utiliza el método *onOptionsItemSelected*, que proporciona Android, para acceder a las pestañas del menú.

```
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case R.id.opciones:
            startActivity(new Intent(this, CCCPreference.class));
            return true;
        case R.id.acercaDe:
            dialogAcercaDe(R.string.acercaDe,R.string.acercaDeHistorium);
            return true;
    }
    return super.onOptionsItemSelected(item);
}
```

Ilustración 37. Método onOptionsItemSelected de la clase SelecciónEpoca.java

5.1.1.6 Módulo relacionado con la interfaz

Se han creado unos ficheros **strings.xml** y **colors.xml**, que contienen los literales y los colores respectivamente utilizados en la aplicación.

También se han creado **buttonshape.xml** y **panel.xml**, que aplican distintos estilos a los botones y paneles respectivamente utilizados en la aplicación.

```
<string name="app_name">Historium</string>
<string name="hello_world">Hello world!</string>
<string name="action_settings">Settings</string>
<string name="initialImage">Imagen de entrada</string>
<string name="comenzar">Comenzar</string>
<string name="seleccionEpoca">Selección de época</string>
<string name="deseaContinuar">¿Desea continuar?</string>
<string name="cancelar">Cancelar</string>
<string name="salir">Salir</string>
<string name="salida">¿Desea salir del juego?</string>
<string name="continuar">Continuar</string>
<string name="epoca">Época</string>
<string name="tablero">Tablero</string>
<string name="eliminar">Eliminar</string>
<string name="borrarBloque">¿Desea borrar este bloque?</string>
```

Ilustración 38. Extracto del fichero strings.xml

5.2 Servidor

En este punto se aportarán algunos datos fundamentales acerca de la implementación que se ha utilizado en el entorno del servidor externo.

5.2.1 Base de datos

Para crear la base de datos bastaba con ejecutar el *script* que creaba las tablas e insertaba los datos. Se utilizó pgAdmin para ejecutar el *script* en la base de datos. Dicho *script* se puede observar en el punto 5.2.3.

Las tablas están descritas en el *Capítulo 4* con el modelo entidad-relación mostrado.

5.2.2 Almacenamiento de imágenes

Para esta aplicación se necesitaba de un espacio reservado para las imágenes que se utilizarían en la misma. Este espacio fue creado en el servidor, en una carpeta que contiene todas las imágenes. El almacenamiento de imágenes se realizó con el programa FileZilla. Si se quiere saber el procedimiento para almacenar estas imágenes, se puede encontrar esta información en el Anexo A.

5.2.3 Script utilizado

El script crea la tabla épocas y texturas donde se almacenarán los datos. También crea la restricción por id de época entre las dos tablas. Una vez que se han creado las tablas y las restricciones, se insertan todos los datos. El Anexo A describe los pasos seguidos para almacenar los datos.

```
CREATE TABLE epocas (
idEpoca SERIAL PRIMARY KEY,
nombreEpoca TEXT UNIQUE,
imagenEpoca TEXT,
descripcionEpoca TEXT);

CREATE TABLE texturas(
idTextura SERIAL PRIMARY KEY,
idEpoca INTEGER,
nombreTextura TEXT,
imagenTextura TEXT);

ALTER TABLE ONLY texturas ADD CONSTRAINT texturas_id_epocas_fkey FOREIGN KEY(idEpoca) REFERENCES epocas(idEpoca);

INSERT INTO epocas (nombreepoca,imagenepoca,descripcionepoca) VALUES ('El antiguo Egipto','http://sacha.ii.uam.es/sergio/imagenes/piramides.jpg'
,'Fue una civilización antigua de la parte oriental de África del Norte, se concentró a lo largo del curso inferior del río Nilo
 en lo que hoy es el estado moderno de Egipto. La civilización se unificó en torno al año 3150 aC ,con la unificación política del Alto
 y el Bajo Egipto en el marco del primer faraón, y desarrollado a lo largo de los próximos tres milenios.');

INSERT INTO epocas (nombreepoca,imagenepoca,descripcionepoca) VALUES ('El Imperio Romano','http://sacha.ii.uam.es/sergio/imagenes/panteon.jpg'
,'Fue una etapa de la civilización romana en la Antigüedad clásica, posterior a la República romana
y caracterizada por una forma de gobierno autocrática. El nacimiento del Imperio viene precedido por la expansión de su capital, Roma,
que extendió su control en torno al mar Mediterráneo.');

INSERT INTO epocas (nombreepoca,imagenepoca,descripcionepoca) VALUES ('La Edad Media','http://sacha.ii.uam.es/sergio/imagenes/castillo.jpg'
,'Fue una etapa de la historia europea que transcurrió desde la desintegración del Imperio romano de Occidente, en el siglo V,
 hasta el siglo XV. Su comienzo se sitúa tradicionalmente en el año 476 con la caída del Imperio Romano de Occidente y
 su fin en 1492 con el descubrimiento de América, o en 1453 con la caída del Imperio Bizantino,
 fecha que coincide con la invención de la imprenta (Biblia de Gutenberg) y con el fin de la Guerra de los Cien Años.');
```

Ilustración 39. Script utilizado para crear la base de datos del servidor

6. PRUEBAS Y RESULTADOS

En este apartado del proyecto se comentarán las pruebas realizadas para comprobar que toda la implementación realizada funciona correctamente. Para seguir un orden concreto se han ido realizando pruebas a medida que se va avanzando por las actividades que forman la aplicación.

Realizar todas estas pruebas no garantiza que la aplicación no tenga ningún error, ya que puede que otro usuario intente un caso que no se haya previsto y obtenga un resultado erróneo. Por eso existe la fase de mantenimiento, donde se podrán observar esos errores que no se habían detectado anteriormente para poder solucionarlos. No obstante, se ha intentado que la aplicación no contenga ningún error.

Se han realizado dos tipos de pruebas típicas en desarrollo de software: pruebas de caja blanca y de caja negra.

6.1 Pruebas de caja blanca

Son llamadas también pruebas de lógica, ya que se refiere a la prueba directa de la lógica de clase. Para realizar estas pruebas, se conoce la implementación del código. De esta manera, se dan distintos valores de entrada y se observa el flujo de ejecución que sigue la aplicación [7].

Estas son algunas de las pruebas de realizadas:

- Comprobación de que se ejecuten todos los bucles en sus límites
- Comprobación tanto de las consultas SQL como de la manera que interactúa con la base de datos.
- Comprobación de la creación de los paneles del modo edición.
- Comprobación del flujo de los algoritmos de movimiento de bloque para que no realice iteraciones innecesarias.
- Pruebas de ejecución con el hilo para que afecte en el rendimiento del juego.

6.2 Pruebas de caja negra

Son llamadas también pruebas de aplicación. Es estudiado desde el punto de vista de las entradas que recibe y las salidas que produce, sin importar el funcionamiento interno de la aplicación. Para estas pruebas no importa el cómo se hace (esto ya se comprobaba en las pruebas de caja blanca), sino que obtengas los resultados esperados.

Estas son algunas de las pruebas de realizadas:

- Comprobar que todos los botones realizan las acciones deseadas.
- Comprobar que se obtienen los resultados esperados por las bases de datos del servidor y del dispositivo.
- Comprobar que las preferencias de la aplicación funcionen como deben.
- Comprobar cómo interactúan los bloques ante los distintos casos.

6.3 Resultados de las pruebas

A continuación, se detallarán las pruebas básicas que se han realizado en la aplicación y los resultados obtenidos:

Descripción de la prueba	**Resultado**
Nombre de usuario y password incorrectos para acceder al servidor.	Devuelve un mensaje que te indica que ha sido imposible conectarse al servidor.
Acceso a una época.	La pantalla contiene los materiales de esa época y tiene todos los paneles necesarios ordenados.
En el **modo edición**, se selecciona una textura diferente a la que está por defecto.	El bloque por defecto cambia a la textura elegida.
Se realiza un corte manual al bloque por defecto.	El bloque por defecto cambia su anchura y se queda con el bloque más grande del corte.
Se cambian las dimensiones del bloque por defecto en los editables. Después, se pulsa el botón "*Cambiar*"	El bloque por defecto cambia su anchura y altura a las medidas impuestas por el usuario.
Se cambian las dimensiones del bloque por defecto en los editables a una anchura de 200 pixeles.	Obtienes un mensaje de error que te indica que el tamaño de la anchura o altura no puede ser mayor de 180 pixeles.
Se pulsa el botón "*Rotar*"	El bloque por defecto rota su posición 90 grados.
Se pulsa el botón "*Crear Bloque*" con algunos cambios realizados en el bloque por defecto.	Se crea el bloque, se añade a la base de datos y se añade al panel de creación de bloques.
Se pulsa el botón "*Deshacer*" con cambios realizados en el bloque por defecto.	El bloque por defecto vuelve a su estado inicial.
Con 8 bloques creados en el panel de creación, se crea un bloque más con el botón "*Crear Bloque*".	El panel de creación de bloques cambia y muestra el último bloque creado con el botón "*Anterior*" para poder acceder a la página anterior.
Se crean 32 bloques en el panel de creación y se utilizan los botones "*Anterior*" y "*Siguiente*" para desplazarse por los bloques creados.	Se pueden observar perfectamente todos los bloques creados en las distintas páginas. También se puede acceder correctamente a ellas con los botones.
Hay 19 bloques creados y mientras que se están observado los 8 primeros bloques, se está editando un bloque nuevo y se pulsa "*Crear Bloque*".	Obtienes un mensaje de error que te indica que debes de estar en la última pestaña del panel de creación para poder crear el bloque.

Se pulsa el botón "*Continuar*" sin tener bloques creados.	Salta un diálogo que te indica que te creará por defecto 25 bloques. Si acepta, pasa a la siguiente pantalla con los bloques creados.
Se pulsa el botón "Continuar" con bloques creados.	Se pasa a la siguiente pantalla con los bloques creados.
En el **tablero de juego**, se pulsa el lienzo.	No coloca ningún bloque.
Seleccionas un bloque y pulsas el lienzo.	Al seleccionar el bloque, se marca con un fondo amarillo y cae por el lienzo desde la posición donde se pulso. Frena cuando detecta la colisión de la pantalla o de otro bloque. Cuando ha realizado el movimiento, la selección pasa a ser roja.
Comprobación de la selección de un bloque con fondo rojo.	No se puede seleccionar un bloque con el fondo rojo.
Tira un bloque de tal manera que su centro de gravedad quede a la derecha de la esquina superior derecha de otro bloque.	Provoca una rotación derecha del bloque lanzado. Realiza una rotación de 90 grados y después cae hasta donde le sea posible.
Tira un bloque de tal manera que su centro de gravedad quede a la izquierda de la esquina superior izquierda de otro bloque.	Provoca una rotación izquierda del bloque lanzado. Realiza una rotación de -90 grados y después cae hasta donde le sea posible.
Se tira un bloque encima de otro de tal manera que no pueda caerse por los lados.	El bloque lanzado se sostiene encima del bloque con el que ha colisionado.
Se tira un bloque encima de otro con el que va a rotar pero al lado del bloque, con el que va a colisionar, hay otro bloque que impedirá una rotación completa.	El bloque rota todo lo que sea posible hasta que colisione con el otro bloque que se lo impide. Con esta situación el usuario debe reiniciar o terminar la partida, ya que no puede colocar más bloques.
Se tira un bloque encima de otro con el que va a rotar pero al lado del bloque, con el que va a colisionar, hay otro bloque que impedirá que rote ya que se sostendrá entre los dos.	El bloque se sostiene entre los dos bloques.
El lienzo tiene una pirámide de bloques y se lanza un bloque desde la cúspide de la pirámide de tal manera que va a rotar con el bloque más alto.	El bloque rota con la cúspide y cuando ha finalizado, detecta que hay otro bloque colocado de la misma manera y comienza otra rotación. Sigue este procedimiento hasta que llega a la base de la pirámide.
Pruebas de rotación con distintos bloques (distinto tamaño y anchura)	Se realizan correctamente las rotaciones y el bloque rotado es el esperado.
Con un bloque en el lienzo, se tira otro bloque de tal manera que no vayan a colisionar entre ellos.	El bloque cae hasta que la pantalla se lo impida, ya que en su trayectoria no se va a encontrar otro bloque.
Se finaliza la partida con una construcción realizada.	Se crea una captura de pantalla de la construcción realizada y se prepara el correo con el nombre del usuario, el correo del profesor y la captura.

Tabla 3. Tabla de pruebas realizadas en el proyecto

7. CONCLUSIONES Y TRABAJO FUTURO

Finalmente, en este punto se comentarán las conclusiones a las que se han llegado tras realizar este proyecto y las posibles mejoras que se realizarán en un futuro.

Para empezar, se han llegado a los objetivos delimitados al principio del proyecto. El objetivo era conseguir una aplicación que sirviese como juego para ser utilizado en los entornos educativos. Desde mi punto de vista, considero que se ha logrado llegar a este objetivo ya que se han dado los primeros pasos para empezar a utilizar este tipo de aplicaciones.

En un primer momento se colaboró con otros profesores de historia que querían hacer uso de la aplicación en sus estudiantes, para analizar los resultados obtenidos y comprobar el uso que hacían de ella. Sin embargo, por cuestiones de tiempo, se decidió declinar ese punto dado que el tiempo para desarrollarla era incompatible con las fechas de los estudiantes. Aun así, me gustaría probar en un futuro esta aplicación con estudiantes de historia para conocer sus opiniones.

En lo personal cabe destacar que este proyecto me ha servido para saber organizar y desarrollar un gran proyecto en un tiempo determinado. Obviamente, me ha servido también para profundizar en mis aptitudes de Android y conocer las distintas características y funcionalidades que permite este sistema.

Fue muy interesante el proceso de aprendizaje, con el que me di cuenta de que poco a poco conseguía llegar a unos resultados más interesantes con los conocimientos que iba adquiriendo. Me habría gustado tener algún desarrollador más y algún diseñador gráfico, que me permitiesen añadir más funcionalidad y más estilos al proyecto. De todos modos, estoy muy satisfecho con el trabajo realizado.

En un futuro me gustaría conseguir una interfaz más llamativa y profesional que consiguiese que el usuario se sintiese atraída por ella. También se debería cambiar algunas estructuras utilizadas en el proyecto, como ya se han indicado en algunos puntos de este documento. Sin embargo, el punto más importante sería implementar un motor de físicas más completo, que reflejase todos los movimientos posibles de los bloques. Por ejemplo, hacer cálculos del centro de gravedad de una construcción para determinar si se caería o no.

Otro punto importante es que el proyecto está implementado para *tablets* con siete pulgadas, dado que la información proporcionada solo se podía mostrar completamente en dispositivos de ese tamaño. Por lo que es lógico que sería necesario un requisito no funcional de portabilidad para que todos los usuarios pudieran utilizarlo en sus dispositivos móviles. Por último, me gustaría utilizar otro servidor diferente dado que ahora se utiliza un servidor compartido con otros estudiantes de la UAM.

8. REFERENCIAS

[1] Xataka, «Y el primer smartphone de la historia fue,» 2001. [En línea]. Available: http://www.xatakamovil.com/movil-y-sociedad/y-el-primer-smartphone-de-la-historia-fue. [Último acceso: 2015].

[2] C. WK, «Calidad del desarrollo software,» 2014. [En línea]. Available: http://confluence.wkcols.com/display/DEV/Quality+on+Software+ Development. [Último acceso: 2015].

[3] Wikipedia, «Wikipedia,» 2012. [En línea]. Available: http://es.wikipedia.org/wiki/Android. [Último acceso: 2015].

[4] P. Molins-Ruano, C. Sevilla, S. Santini, P. Haya y G. Sacha, «Designing videogames to improve students' motivation,» *Elsevier*, 2013.

[5] Synergix, «Tipos de requisitos: Funcional vs No Funcional,» [En línea]. Available: https://synergix.wordpress.com/2008/07/07/requisito-funcional-y-no-funcional/. [Último acceso: 2015].

[6] Wikipedia, «SQLite,» 2012. [En línea]. Available: http://es.wikipedia.org/wiki/SQLite. [Último acceso: 2015].

[7] O. Jiménez y S. Rincón, «Pruebas de caja blanca y negra,» 2012. [En línea]. Available: http://es.slideshare.net/rinconsete/pruebas-de-caja-blanca-y-negra. [Último acceso: 2015].

[8] Filezilla, «Filezilla,» [En línea]. Available: https://filezilla-project.org/download.php?type=client . [Último acceso: 2015].

[9] PgAdmin, «PgAdmin,» [En línea]. Available: www.pgadmin.org/download/. [Último acceso: 2015].

ANEXO A: MANUAL PARA EL PROFESOR

Los profesores pueden tener la necesidad de actualizar la aplicación introduciendo nuevas épocas y texturas. Al no haber podido construir una página web, que contuviese un formulario con campos a rellenar para poder introducir la información mencionada anteriormente, el profesor deberá introducir los datos manualmente en la base de datos y servidor. Aunque el profesor no tenga conocimientos amplios en informática, con este manual sabrá introducir fácilmente todos los datos que necesite.

En primer lugar, el profesor debe tener instalado en su ordenador pgAdmin, para introducir la información en la base de datos, y Filezilla, para introducir las imágenes en el servidor.

En [8] puedes descargarte el cliente FTP Filezilla. Mientras que en [9] podrás descargar la herramienta pgAdmin.

Configuración de las herramientas

PgAdmin

Una vez que se ha descargado la herramienta, la abrimos para empezar a configurarla.

Para poder añadir una conexión al servidor, debemos pulsar en el icono del enchufe que pone "*Add a connection to a server*". Cuando lo hayamos pulsado, obtendremos una ventana para registrar los datos referentes al servidor. Los datos que debemos meter son los siguientes:

- Name: sacha.ii.uam.es
- Host: sacha.ii.uam.es
- Port: 5432
- Maintenance: postgres
- Username: (Usuario proporcionado al profesor)
- Password: (Password proporcionado al profesor)

Con estos datos, la pantalla debería quedar como en la *Ilustración 40*.

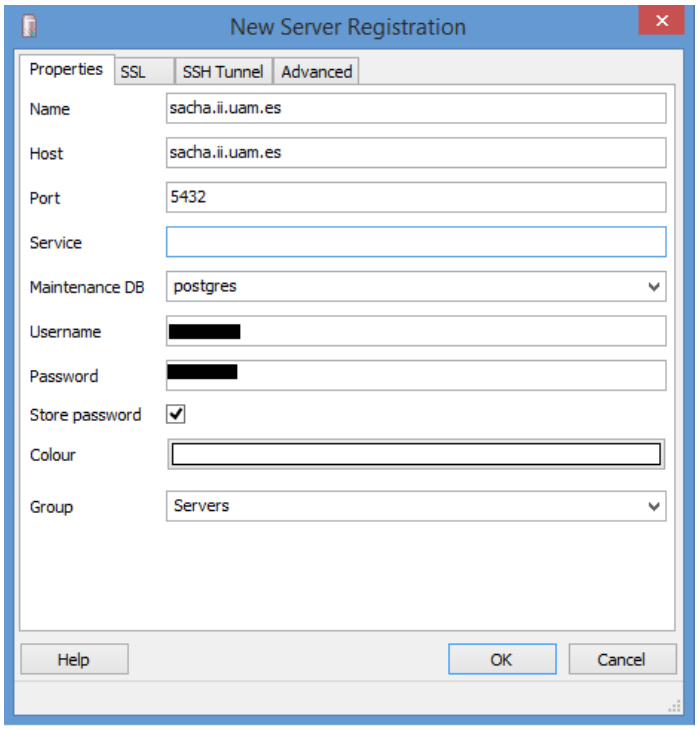

Ilustración 40. Ventana de configuración PgAdmin

Pulsamos OK y ya estaremos conectados a las bases de datos del servidor. Al tratarse de un servidor compartido, hay varias bases de datos que están siendo utilizadas por otras personas. Para este proyecto se ha utilizado la base de datos llamada "sergioTFG". Por lo que si pulsamos en la carpeta "sergioTFG", estaremos ya en la base de datos. Como se puede observar en la *Ilustración 41*, a través de "Schemas"- "Public" - "Tables", podemos ver las dos tablas creadas para la aplicación: épocas y texturas.

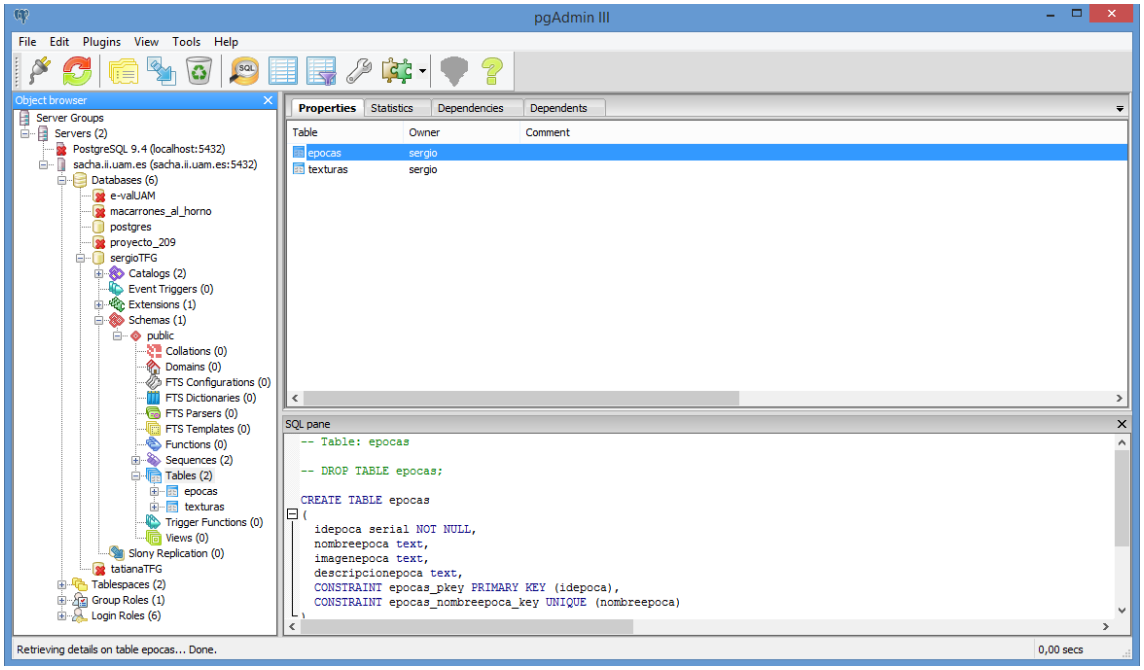

Ilustración 41. Base de datos "sergioTFG"

Para ver los datos introducidos en las tablas, basta con seleccionar una de las tablas y pulsar el botón superior que tiene forma de tabla y que pone "*View the data in the selected object*". Si seleccionamos la tabla de épocas, por ejemplo, y pulsamos el botón, obtendremos la imagen de la *Ilustración 42*.

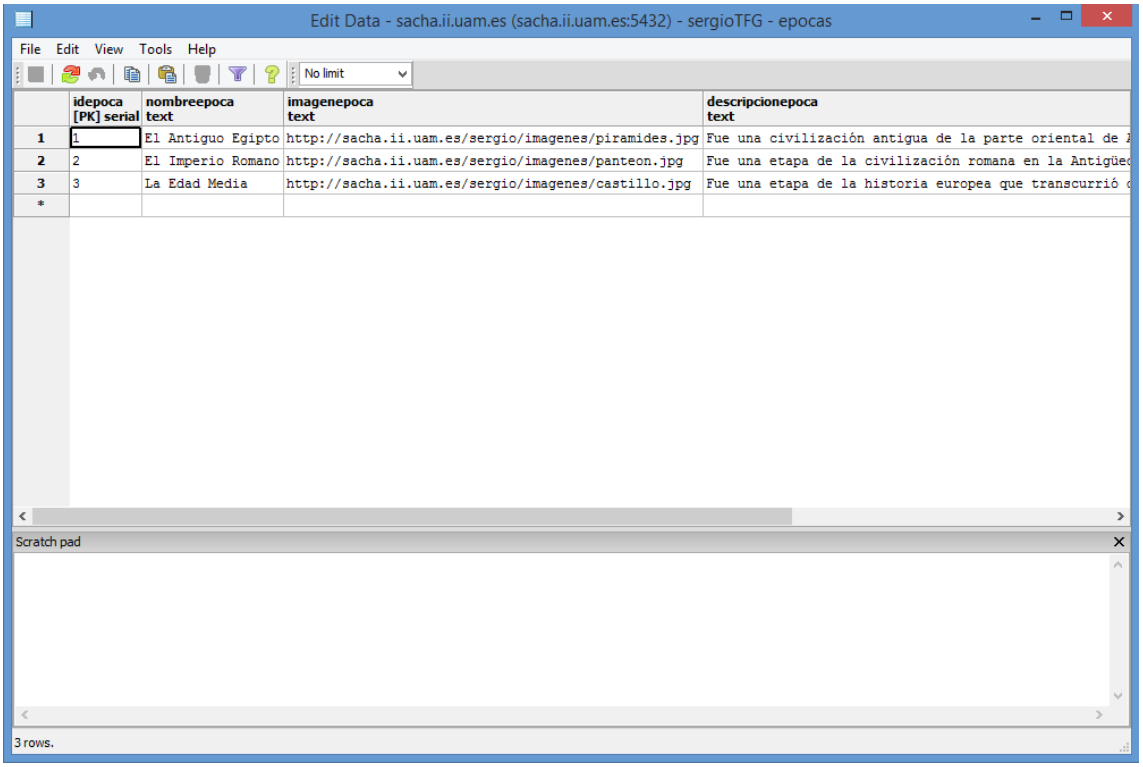

Ilustración 42. Datos de la tabla épocas

Filezilla

Ahora pasamos a configurar el cliente FTP Filezilla. Una vez que tenemos instalado la herramienta, la abrimos y nos encontraremos una ventana con distintas secciones. Este programa es más fácil que pgAdmin, ya que simplemente para conectarnos basta con que rellenemos los datos de la parte superior con los siguientes:

- Servidor: sacha.ii.uam.es
- Nombre de usuario: (Usuario asignado al profesor)
- Password: (Password asignada al profesor)
- Puerto: 21

Una vez que tenemos los datos metidos, pulsamos "*Conexión rápida*" y obtendremos la imagen de la *Ilustración 43.*

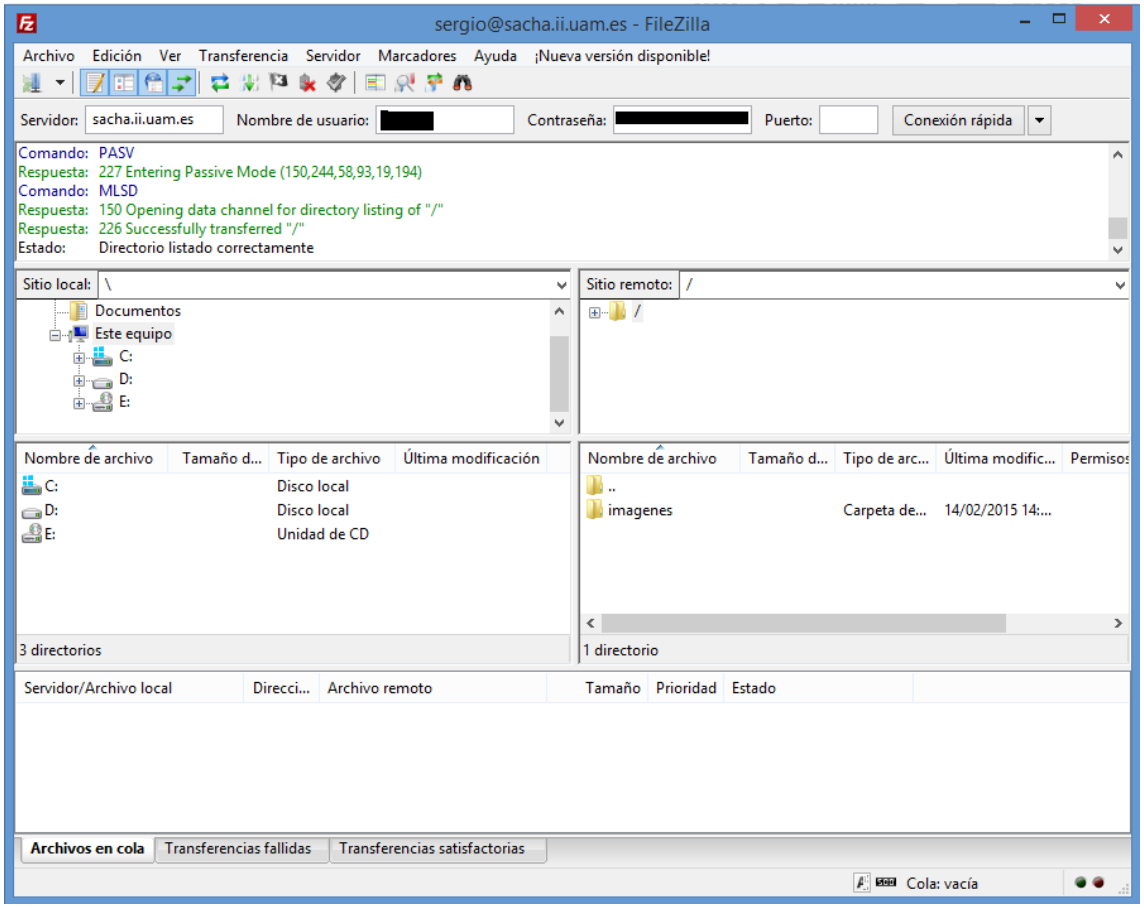

Ilustración 43. Contenido del servidor

En la ventana derecha podemos ver el contenido del servidor. De esta manera observamos como tenemos la carpeta "*imágenes*", donde van introducidas todas las imágenes que se utilizarán en la aplicación.

Como introducir datos en el servidor

Ahora que tenemos configuradas las herramientas, procedemos a introducir nuevos datos. Tomaremos como ejemplo que el profesor desea introducir una época y una textura perteneciente a esa época.

Los primeros pasos a seguir son buscar las imágenes que se relacionarán a la época y al material escogidos. Una vez que tenemos las dos imágenes que vamos a utilizar, primero deben introducirse en las imágenes del servidor. Para ello nos conectamos al servidor con Filezilla, como ya se ha comentado en el punto anterior. Para introducir la imagen en el servidor, basta con arrastrar las imágenes a la carpeta "*imágenes*", que está en la ventana derecha del programa.

Ahora tenemos las imágenes en el servidor, solo falta añadir la información correspondiente a la base de datos. Nos conectamos a través de pgAdmin, con los datos formulados anteriormente, y accedemos a la base de datos "sergioTFG".

A continuación, tendremos que introducir un par de líneas SQL para introducir la época y la textura. Estando en “sergioTFG”, pulsamos un botón superior con forma de lupa y que contiene la cadena “SQL”. Se nos abrirá otra ventana donde deberemos escribir las líneas de código.

Pongamos que queremos introducir como etapa la edad actual y como textura el hormigón.

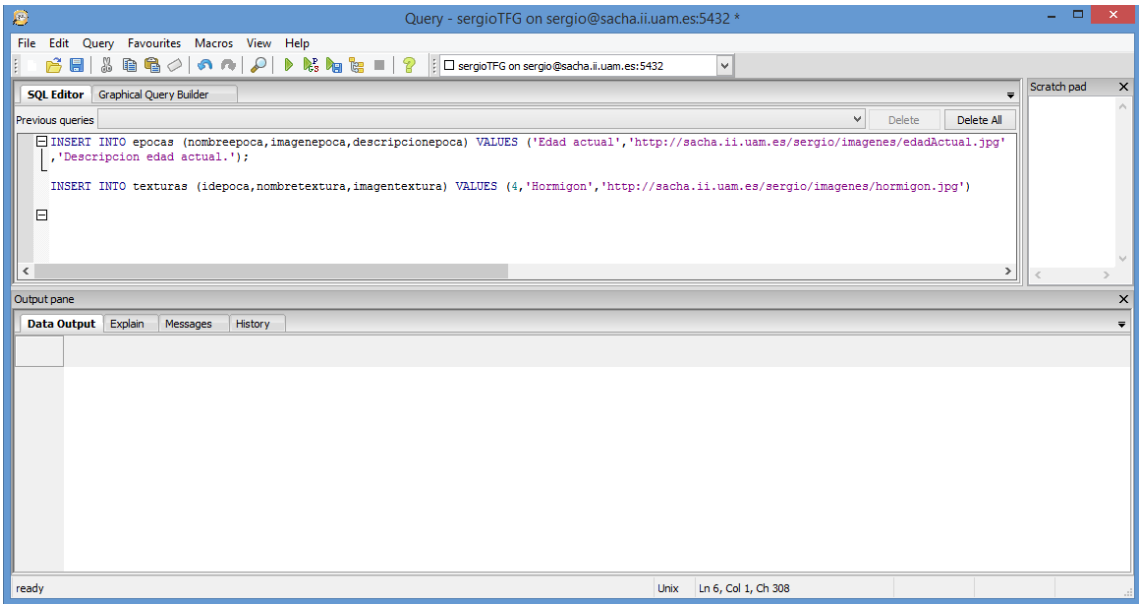

Ilustración 44. Líneas SQL de inserción de época y textura

Como se puede apreciar en la *Ilustración 44*, debemos realizar un *insert* para cada una de las épocas:

```
1.

INSERT INTO epocas (nombreepoca,imagenepoca,descripcionepoca) VALUES
('Edadactual','http://sacha.ii.uam.es/sergio/imagenes/edadActual.jpg',
'Descripcion edad actual.');

2.

INSERT INTO texturas (idepoca,nombretextura,imagentextura) VALUES
(4,'Hormigon','http://sacha.ii.uam.es/sergio/imagenes/hormigon.jpg')
```

Para introducir la época, deberá realizar una inserción como la 1. El profesor únicamente debería cambiar el segundo paréntesis para las próximas épocas, donde la primera cadena es el nombre de la época, la segunda la dirección donde se encuentra la imagen de la época en el servidor y la tercera la descripción de la época.

Para introducir la textura, deberá realizar una inserción como la 2. El profesor únicamente debería cambiar el segundo paréntesis para las próximas texturas, donde la primera cadena es el id de la época donde lo va a introducir (puede sacar este dato observando la tabla de épocas), la segunda el nombre de la textura y la tercera la dirección donde se encuentra la imagen de la textura en el servidor.

Es importante el orden, primero la época y luego la textura porque si no las inserciones serán incorrectas. Finalmente, para ejecutar las líneas SQL habrá que pulsar el botón del triángulo verde y ya tendremos los datos en nuestra base de datos.

ANEXO B: MANUAL DE USUARIO

En este punto se indicarán los pasos que debe seguir el usuario para disfrutar de todas las posibilidades que ofrece la aplicación.

En primer lugar, cuando cargamos la aplicación obtendremos la imagen de la *Ilustración 45*.

Ilustración 45. Pantalla Inicial

Únicamente con pulsar la pantalla, pasaremos a la siguiente (*Ilustración 46*).

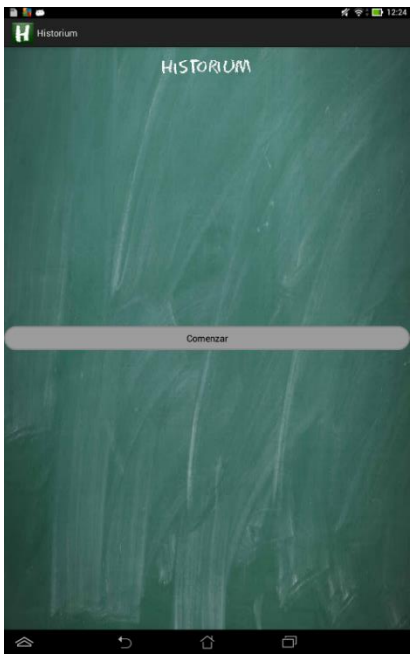

Ilustración 46. Menú Inicial

Esta pantalla contiene un único botón que te permite acceder al juego. Si pulsas el botón, podrás acceder a la pantalla de selección de época.

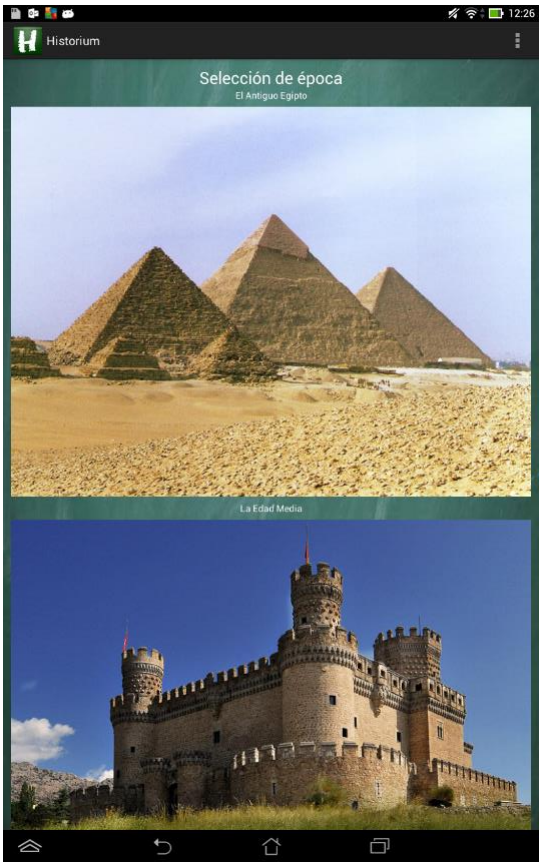

Ilustración 47. Selección de época

En la *Ilustración 47* se puede visualizar la pantalla que contiene las épocas disponibles.

Las épocas se irán cargando según se vaya obteniendo la información del servidor. Dado que en la pantalla solo se pueden ver dos de las tres épocas que contiene la base de datos, basta con desplazar el dedo hacia arriba para visualizar las épocas restantes.

Si seleccionamos una época, observaremos la descripción de la misma (*Ilustración 48*).

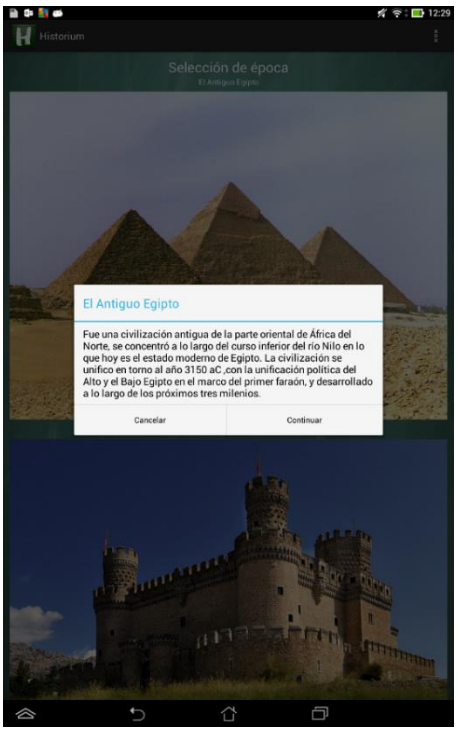

Ilustración 48. Diálogo de Selección de época

Si pulsamos el botón "*Continuar*", pasamos a la pantalla de edición de bloques (*Ilustración 49*).

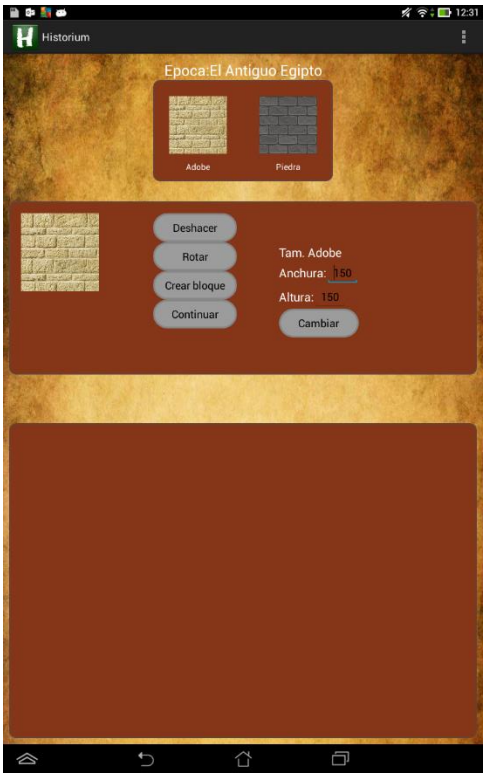

Ilustración 49. Modo edición de bloques

Esta pantalla tiene una sección donde están las texturas disponibles de las épocas, una sección de edición con distintos botones para editar el bloque y una última sección, donde se situarán los bloques creados.

Si se pulsa el botón "*Deshacer*", los cambios hechos volverán al estado inicial. El botón "Rotar", rotará el bloque y con "*Crear Bloque*", se creará el bloque y se pondrá en la sección de bloques creados.

Se pueden cambiar los pixeles de los editables de "*Anchura*" y "*Altura*", y pulsar "*Cambiar*" para actualizar los datos. También se puede realizar el corte manual trazando una línea vertical de arriba abajo en el bloque de edición (siempre que este activado en las preferencias).

En la *Ilustración 50*, se puede observar un ejemplo de creación de distintos bloques.

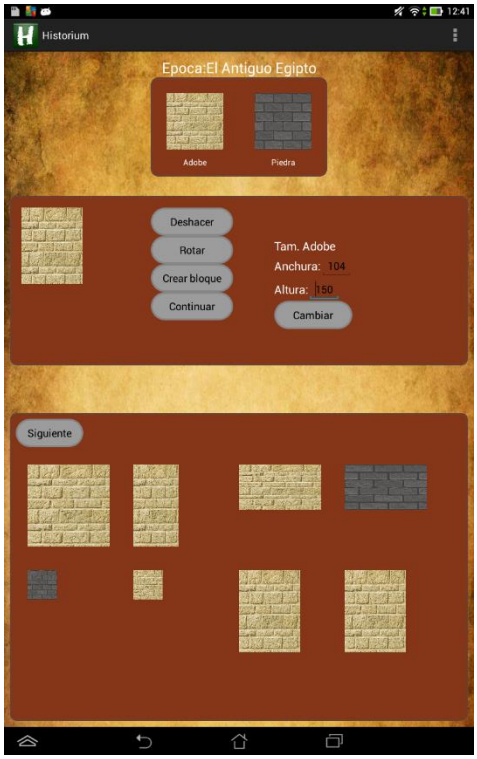

Ilustración 50. Modo edición de bloques con bloques creados

La sección de bloques creados tiene botones con los que poder desplazarse en caso de que se hayan creado múltiples bloques.

Cuando el usuario lo desee, puede pasar a la pantalla de juego pulsando el botón "*Continuar*" (si el usuario no tiene bloques creados, la aplicación le ofrecerá la posibilidad de crear bloques por defecto).

Una vez que el usuario está en la pantalla de juego, tiene todos los bloques creados en el panel superior. El usuario escoge el bloque que desee y después pulsa en la pantalla en el lugar donde desea que caiga. A continuación, el bloque que ha descendido no podrá ser seleccionado más (sino se tiene elegida la opción de bloques infinitos). Todos estos datos se pueden ver en el ejemplo de la *Ilustración 51*, que tiene los bloques creados de la *Ilustración 50*.

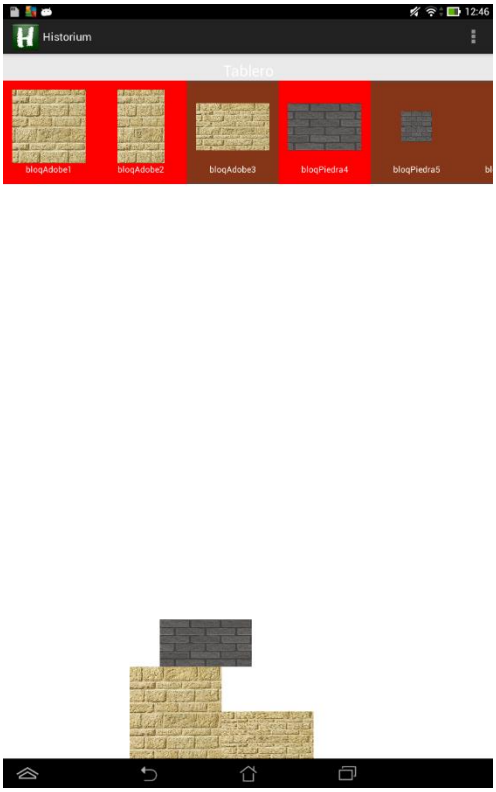

Ilustración 51. Tablero de juego con algunos bloques colocados

Cuando el usuario lo desee, podrá pulsar el menú y acceder a la opción "*Finalizar Partida*" para acabar su construcción y enviarse por correo, como se puede ver en la *Ilustración 52*.

Ilustración 52. Correo preparado para enviar al profesor

Es posible acceder a las preferencias de la aplicación desde la pantalla de Selección de época y el modo edición de bloques. Basta con pulsar el botón menú, desde cualquiera de las dos pantallas indicadas, y seleccionar "*Opciones*" para ir a las preferencias. Las preferencias se pueden observar en la *Ilustración 53*.

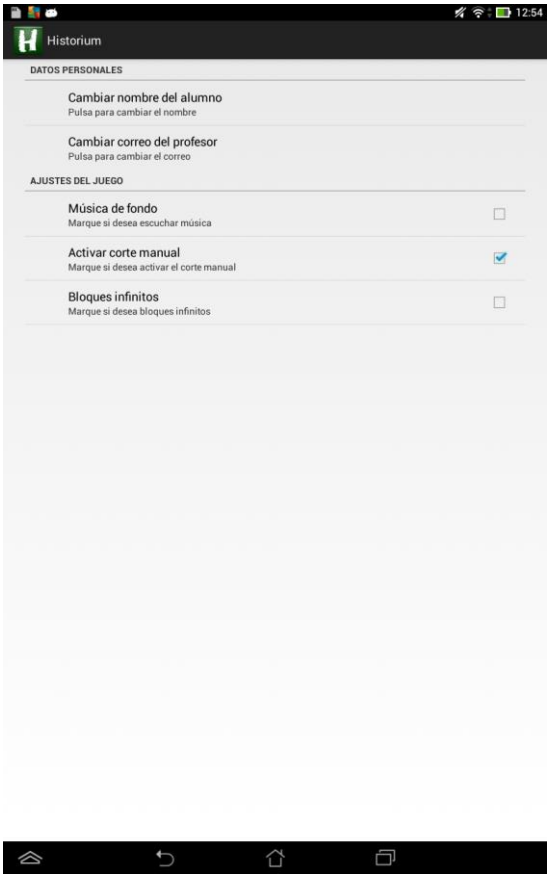

Ilustración 53. Menú de preferencias

Por último, si se desea saber más información de la aplicación, se puede acceder desde el menú y con la opción "*Acerca de*" (desde Selección de época y el modo edición de bloques) obtendrá un dialogo como el que se muestra en la *Ilustración 54*.

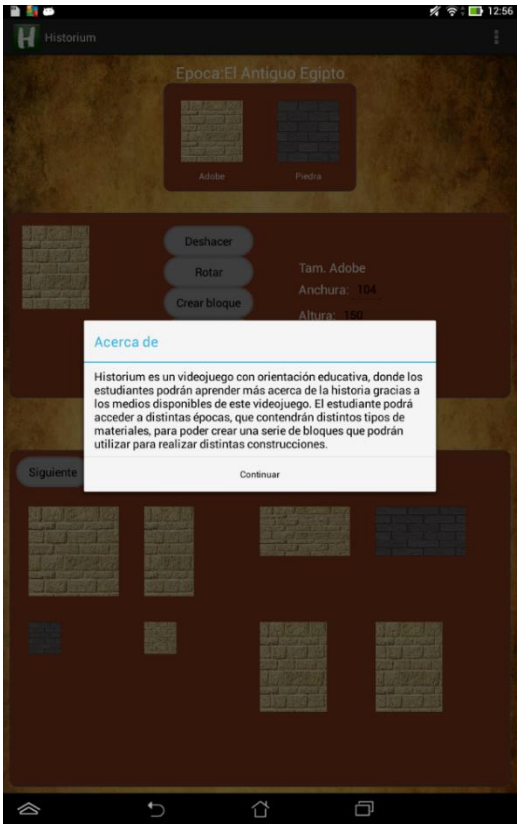

Ilustración 54. Diálogo de "Acerca de"